# UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Escuela Técnica Superior de Ingeniería Industrial

DEPARTAMENTO DE INGENIERÍA DE SISTEMAS Y AUTOMÁTICA


# Control cinemático y de fuerza de una mano robótica para el agarre estable

**TRABAJO FIN DE GRADO**

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA


**Autor:** **José Luis Samper Escudero**

Director: Jorge Feliu Batlle

Codirector: Carlos Alberto Díaz Hernández



Cartagena, Septiembre 2014

*A mi familia, especialmente a mis padres, por todo su apoyo, preocupación y no haber dejado de creer en mí. A ti Abuelo, por toda tu preocupación al ver cómo ha sido este verano.*

*A José Ángel, por su amistad y apoyo durante todos estos años.*

*A Macanás, por haberme embarcado en este proyecto y compartir numerosos momentos de crispación y locura.*

*Al grupo de, ahora sí, ingenieros que tan duro hemos trabajado estos cuatro años para lograr lo que queríamos.*

*A los investigadores de la ROS-RM y al conjunto de profesores y profesionales que he conocido estos años, por transmitir sus motivaciones y aspiraciones, buenas o malas.*

*A Manu, por aparecer en el último momento y recordarme por qué entré en la carrera.*

# Índice.

# CAPÍTULO 1.
# MOTIVACIONES Y OBJETIVOS.

## 1.1. Antecedentes.

A lo largo de la carrera han sido numerosas las ocasiones en las que hemos escuchado hablar de robots. Posiblemente las clases de D. Juan López Coronado, respaldas por las elaboradas prácticas de D. Jorge Feliu Batlle, supusieron el primer acercamiento de forma directa a los sistemas robotizados.

Dichos docentes no sólo presentaron los fundamentos de la cinemática, sino que en contadas ocasiones trataron de despertar el interés de sus alumnos hacia este campo. Para ello, se sirvieron de anécdotas propias, curiosidades aprendidas en congresos recientes y presentación de los últimos hitos de la robótica; desde conceptos como el desarrollo de exoesqueletos controlados mentalmente hasta drones, pasando por los brazos robóticos industriales que todos hemos conocido. Este interés, unido a la curiosidad por la robótica que culminó en la selección de este grado y alimentado por aportaciones puntuales del resto de docentes del departamento de sistemas avanzados y automática, propiciaron que el desarrollo de una aplicación robótica constituyese una de las opciones temáticas sobre las que realizar el presente Trabajo de Fin de Grado.

Dado que el campo de la robótica se encuentra en continuo desarrollo y descubre un abanico de posibilidades vertiginoso, resulta necesario analizar cada una de las posibilidades con detenimiento. Es en este punto cuando, junto a un compañero, Joaquín Macanas Valera; se decide acudir a D. Jorge Feliu Batlle en busca de asesoramiento y directrices. Tras modificar ciertos detalles puntuales de las ideas propuestas por los alumnos, quedó definido el proyecto y los objetivos principales a cumplir los cuales han sido ampliados a lo largo del desarrollo.

## 1.2. Motivaciones.

De entre el conjunto de motivaciones que pueden ser citadas para justificar la elección del presente proyecto, es necesario mencionar, en primer lugar, el auge que experimenta la robótica actualmente. Diariamente somos sorprendidos por nuevos desarrollos e ideas que hasta no hace muchos años habrían otorgado la "medalla de soñador del siglo" a su pensador, un claro ejemplo lo constituye el nuevo concepto de entrega de pedidos planteado por la empresa Amazon para un futuro no muy lejano; uso de drones. Otro clara muestra de ello se puede encontrar en la conferencia sobre RASP que tuvo lugar en la UPCT y a la que, por suerte y gracias a D. José Luis Muñoz Lozano, pude asistir. Por tanto, siendo consciente del despegue que presenta un cierto campo tecnológico ¿Por qué no profundizar en una tecnología que presenta mayor velocidad de desarrollo que desarrolladores?

Una segunda motivación se sitúa en las oportunidades y aplicaciones que ofrece la robótica, usada de un modo adecuado y responsable. Es frecuente, al hablar de robots a conocidos y amigos, que las imágenes de "terminator", "yo, robot" y ficción similar crucen su mente. En otras ocasiones, simplemente les asola la idea de que "estamos eliminando puestos de trabajo", no concibiendo mayor aplicación práctica para la robótica, fuera de la industria, que los robots móviles y, como suele decirse, "amos de casa". No obstante, y cada vez más frecuentemente, vemos el giro de 180 grados que es capaz de infundir la robótica en la vida de una persona con discapacidades. Bajo estos términos trato de referirme a la utilidad que presentan los exoesqueletos, manos robóticas controladas por contracción muscular y "globos oculares robóticos" adheridos al sistema nervioso, entre otros. Se plantea así el segundo interrogante que bien puede actuar de catalizador ¿Por qué no aprender más sobre una tecnología capaz de devolver parte de la calidad de vida de que gozaban personas con limitaciones?

El siguiente motivo se localiza en la curiosidad y admiración que despiertan los sistemas robóticos. Así, como en el cambio de perspectiva y forma de interpretar el entorno que otorga la robótica. A lo largo del proyecto, han sido numerosas las ocasiones plagadas de desesperación, de intentos fallidos y replanteamiento de motivaciones que han sido afrontadas. Muchas de estas se han salvado con perseverancia, aunque otras han requerido replantear los problemas y afrontar la realidad desde una perspectiva distinta nunca antes considerada, véase, por ejemplo, las complejas relaciones que establecemos de forma natural al tratar de agarrar un objeto; tan fáciles de realizar para un humano común y de complejidad inigualable cuando se trata de "explicar" a un robot cómo hacerlo.

Por último, y no por ello de menor importancia, la idea de contribuir y "participar", en cierto modo, a la realización de un proyecto europeo supone un gran aliciente.

## 1.3. Objetivos.

Este proyecto tratar de allanar el camino para futuros desarrollos que involucren alguno de los pilares fundamentales del mismo; detección, posicionamiento y reconocimiento de objetos por medio de Point Cloud Library, uso de la mano Shadow Hand UPCT, uso de ROS y sincronización entre dos robots. Así como la comunicación entre ROS y un robot no embebido en dicho sistema por incompatibilidades.

De este modo, el proyecto trata de servir a fines mayores; por un lado, puede definirse como parte de un proyecto de mayor envergadura concebido junto a J.Macanás: Sincronización entre el robot Shadow Hand y un brazo robótico LWA-4P de Robotnik para alcanzar y agarrar de forma estable diversos objetos. Por otro lado, trata de servir como herramienta de iniciación o apoyo para el grupo NEUROCOR en la realización de su proyecto internacional relacionado con el desarrollo de un exoesqueleto.

Así, sin más preámbulos, se puede señalar que este proyecto presenta como objetivos primordiales:

- Aprendizaje y familiarización con Linux (Debian **y Ubuntu**).
- Puesta en marcha del robot Shadow Hand.
- Desarrollo de software básico de movimiento del robot usando API's y C básico.
- **Aprendizaje y familiarización con ROS.**
- **Aprendizaje de C++ y familiarización con Python.**
- **Integración, configuración y prueba del robot en ROS.**
- Realizar agarres estables de objetos con el robot Shadow Hand.
- **Desarrollo de librerías propias de control en C++.**
- **Desarrollo de un sistema de visión artificial con reconocimiento de objetos y triangulación de coordenadas.**
- **Puesta en marcha del sistema de Visión Artificial.**
- **Reconocer los objetos a agarrar mediante un sistema de visión artificial integrado en ROS.**
- **Adhesión física del robot Shadow Hand al brazo Robotnik Schunk LWA-4P.**

- Desarrollo de las pautas para la sincronización del robot con un brazo robótico, programado por terceros.

Los objetivos se han presentado desglosados según las metas puntuales definidas para cada una de las fases en las que se ha estructurado el proyecto. Los elementos remarcados constituyen partes adicionales, no incluidas en la rúbrica, que complementan los objetivos principales, añadiendo valor al desarrollo realizado.

Para concluir, se muestra un esquema del sistema completo que pretende desarrollarse al unir este proyecto al de control de un brazo robótico desarrollado por Joaquín Macanás Valera.

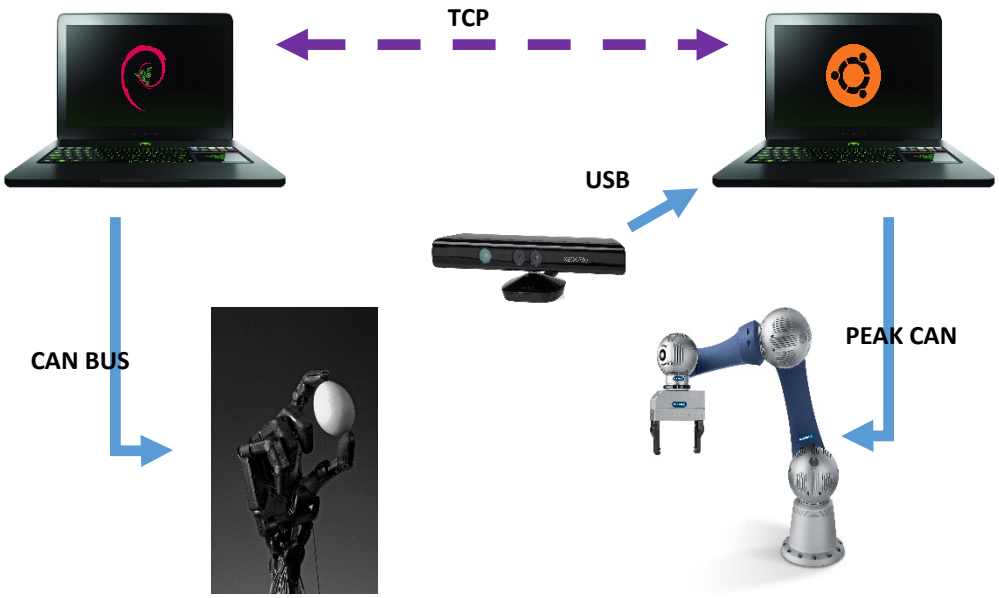



*Ilustración 1. Objetivo Global del proyecto.*

## 1.4. Descripción de los capítulos.

Capítulo 1: Motivaciones y Objetivos.

En este capítulo se describirán los objetivos que se buscan alcanzar con la realización del proyecto y los temas que se van a tratar en cada capítulo.

Capítulo 2: Estado del arte: Robótica y Robots Manipuladores.

Este capítulo establece el contexto en el que puede quedar embebido este TFG dentro del marco de la robótica actual.

Capítulo 3: Software Aplicado a la Robótica.

Este capítulo puede considerarse una segunda parte del estado del arte encauzada a debatir y presentar las distintas opciones software aplicables al proyecto.

Capítulo 4: Definición de los elementos Hardware del proyecto.

En este capítulo se presentan los distintos elementos hardware de que consta el proyecto.

Capítulo 5: Configuración del Software y primeros pasos.

A lo largo del capítulo se definen con mayor profundidad los elementos software empleados, así como se proporcionan pautas y guías para la configuración y uso de estos sistemas.

Capítulo 6: Software Desarrollado.

Este capítulo, junto al anterior, define las aportaciones que realiza el proyecto. A lo largo de esta sección se enumeran y justifican las distintas soluciones desarrolladas a nivel software.

Capítulo 7: Hardware Desarrollado.

Este título hace referencia a los elementos hardware que fue necesario desarrollar para completar el desarrollo global del que forma parte este proyecto.

Capítulo 8: Futuros Desarrollos.

Esta sección propone un conjunto de posibles trabajos y desarrollos que pueden realizarse para mejorar el diseño actual.

# CAPÍTULO 2

# ESTADO DEL ARTE:

## ROBÓTICA Y ROBOTS MANIPULADORES.

## 2.1. Introducción: ¿Qué es la robótica?

Conviene comenzar este capítulo definiendo qué es la robótica. Según la Real Academia de la Lengua Española, ésta puede quedar definida como:

> *"Técnica que aplica la informática al diseño y empleo de aparatos que, en sustitución de personas, realizan operaciones o trabajos, por lo general en instalaciones industriales."*

Aunque válida, esta definición puede resultar un tanto ambigua. Ciertos autores, redefinen el término de la siguiente forma:

> *"Dícese de la ciencia y tecnología encargada del diseño, manufactura y aplicación de los robots"*

De esta segunda interpretación surge una pregunta que aunque sencilla, dada la extensión de su definición, puede acarrear algún que otro quebradero de cabeza al tratar de formular su solución con el mayor grado de acierto posible: ¿Qué es un robot?

Si formulásemos dicha pregunta a distintas personas, especialistas o no, comprobaríamos rápidamente que; aunque similares, todas las respuestas serían distintas, pudiendo llegar a plantearse incompatibilidades entre ellas. A continuación, se citan ciertas definiciones del término con diversa validez:

> *"Manipulador funcional reprogramable, capaz de mover material, piezas herramientas o dispositivos especializados mediante movimientos variables programados, con el fin de realizar tareas diversas" (Robot Industries Association).*

> *"Conexión inteligente de la percepción y la acción" (Isaac Asimov).*

> *"Dispositivo humanoide con cierto grado de inteligencia, que substituye a las personas en la realización de tareas útiles" (definición popular).*

> *"Máquina o ingenio electrónico programable, capaz de manipular objetos y realizar operaciones antes reservadas sólo a las personas"*

Los ejemplos anteriores tratan de evidenciar el grado de incertidumbre y desacuerdo que puedo llegar a existir al tratar de definir un término que "a priori" podría considerarse evidente. Este hecho puede justificarse a partir de la amplitud del citado campo así como del desarrollo exponencial que ha experimentado la robótica durante las últimas décadas. Salvando las incompatibilidades o limitaciones inherentes a la especialización del entrevistado, es posible observar un segundo foco de incertidumbre procedente de la ciencia ficción y el papel que ha jugado ésta en la aceptación de los robots por la sociedad.

Tal y como se expondrá en el siguiente punto, la ciencia ficción ha jugado un papel fundamental a la hora de presentar la robótica a la sociedad. Un claro y curioso ejemplo de ello lo constituye la acuñación del término robot, este vocablo surgió en 1921 de manos del escritor Karel Capek al estrenar éste su obra "Rossum's Universal Robot" en el teatro nacional de Praga.

## 2.2. Breve Repaso Histórico.

### 2.2.1. Precursores de los Robots.

A diferencia de otras disciplinas, la situación de la robótica en un contexto histórico resulta una tarea ardua dadas las discrepancias existentes en cuanto a la definición de la materia. Encontramos así, principalmente, dos teorías o posturas que tratan de definir el marco histórico de dicha disciplina. Por un lado se encuentran los defensores de George Devol, desarrollador del primer robot industrial en el año 1921, como pilar de la robótica. En contraposición, se ubican aquellos autores que consideran la robótica una evolución natural de la automática.

Con objeto de abarcar un mayor rango contextual y considerando la elección del marco, dados los antecedentes, un hecho con ciertos matices personales, se desarrollarán los antecedentes desde el punto de vista evolucionista.

A lo largo de su existencia, el ser humano ha tratado de construir máquinas y artefactos que faciliten su trabajo o permitan satisfacer sus necesidades y deseos. Evidencia de ello lo constituyen los brazos mecánicos presentes en estatuas egipcias que trataban de lograr el movimiento automático

*Ilustración 2.Robot de Leonardo*

de los mismos. Con el devenir de los tiempos, la frecuencia y calidad de estos desarrollos se ha mejorado. Entre ellos puede destacarse el denominado "Robot de Leonardo" desarrollado por Leonardo Da Vinci. Este pensador destacó por realizar los bocetos y planos necesarios para realizar un mecanismo de figura antropomórfica, similar a la de un guerrero, capaz de levantarse y sentarse de forma automática.

No obstante, aunque todos estos desarrollos permiten realizar ciertas tareas de forma automática, no pueden ser considerados autómatas como tal ya que, entre otros aspectos, carecen de una de las características básicas para ello: ser programables.

### 2.2.2. Primeros Robots.

Para la aparición del primer autómata es necesario esperar hasta mediados del siglo XVIII, momento en el que acaece una revolución en la industria textil impulsada por el maestro tejedor Joseph-Marie Jacquard. Este mecánico e inventor, originario de Lyon, realizó una máquina textil programable mediante tarjetas perforadas. La máquina de Jacquard, permitía fabricar telas con hilos de distintos colores y complicados dibujos. Para ello constaba de un paquete de tarjetas de cartón perforadas, en el cual se especificaba el patrón a tejer, que eran cambiadas mediante la acción de un pedal. A este ingenio le siguieron otros sistemas, basados en las ideas de Jacquard, que trataban de automatizar otra serie de procesos. Entre ellos, debe destacarse el primer robot industrial desarrollado por George Devol y patentado en 1952.

George Charles Devol, Jr, inventor estadounidense nacido en Louisville, Kenctucky, con más de 40 patentes a sus espaldas y presidente de la empresa *Devol Research*, es considerado por muchos el "padre de la robótica". Y es que, entre las numerosas patentes que figuran a nombre de Devol, destaca, con especial importancia para este campo en concreto, la del primer brazo robótico programable operado digitalmente.

Este robot, conocido con el apelativo *"Unimate"*, marcó dos hitos en la historia. En primer lugar, se convirtió en el primer robot comercializado. Por otro lado, constituyó el primer robot industrial instalado y totalmente funcional. Como dato curioso, señalar que la primera instalación y venta se realizó, en 1961, en una cadena de montaje de *General Motors*, concretamente en la *Inland Fisher Guide Plant* situada en Nueva Jersey. Específicamente, el citado robot se encargaba de transportar las piezas fundidas en molde hasta la cadena de montaje, dónde posteriormente soldaba dichas partes al chasis del vehículo en producción.

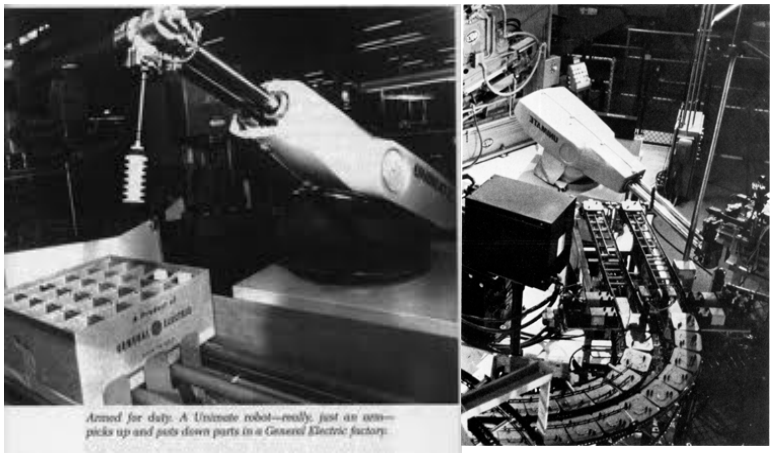

Armed for duty. A Unimate robot—really, just an arm—picks up and puts down parts in a General Electric factory.

*Ilustración 3. Robot Unimate*

Como puede comprobarse en las imágenes, el robot Unimate constaba de una gran caja, en la que se realizaban los cómputos, unida a un segundo cajón, encargado de realizar el conexionado con el brazo articulado. Por último, como dato de interés, mencionar que dicho robot fue admitido en el Robot Hall of Fame en el año 2003.

Muchos autores consideran que la instalación de Unimate en General Motors marcó un punto de inflexión en el desarrollo de la robótica. Tanto es así que, en 1968, J.F. Engelberger, colaborador de G. Devol, visitó Japón y poco más tarde se firmaron acuerdos con Kawasaki para la construcción de robots tipo Unimate. El crecimiento de la robótica en Japón aventajó en breve a los Estados Unidos gracias a Nissan, que formó la primera asociación robótica del mundo, Asociación de Robótica industrial de Japón (JIRA), en 1972. Tan sólo dos años más tarde se formaría el Instituto de Robótica de América (RIA), que en 1984 cambió su nombre por el de Asociación de Industrias Robóticas, manteniendo las mismas siglas. Por otra parte, Europa tuvo un despertar más tardío. Ya que no sería hasta 1973, cuando la firma sueca ASEA construyese el primer robot con accionamiento totalmente eléctrico.

Tal y como predijo G. Devol, a partir de este momento, se difundió el uso de robots en la industria con objeto de mejorar la calidad, el rendimiento y la seguridad de los procesos. El siguiente punto de avance se puede localizar en el año 1973 con el desarrollo del primer robot con seis articulaciones electromecánicas desarrollado por KUKA Robot Group con el nombre "Famulus". Según la propia empresa, el producto se desarrolló con objeto de cubrir las necesidades de potencia y seguridad demandadas por la industria automovilística de la época.

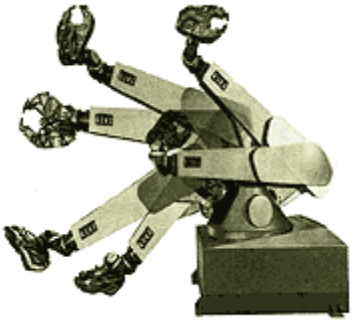


*Ilustración 4.Robot Fumulus*

Con la entrada de los años 90, el desarrollo de nuevos robots y prototipos alcanzó valores máximos los cuales han ido incrementando exponencialmente hasta la actualidad. Sin embargo, dado el elevado número de apariciones, no es posible hacer referencia a todas ellas en el presente documento por cuestiones de objetivos y extensión. A pesar de ello, en los puntos venideros se mencionarán algunos ejemplos de interés para la aplicación.

Hasta el momento solamente se han mencionados aspectos relativos a desarrollos hardware y software. Sin embargo, en puntos anteriores se enunció que los medios de comunicación y la ciencia ficción jugaron un papel de especial importancia en el desarrollo de la robótica. Aunque se acostumbra a dejar de lado, al abordarse temas ingenieriles, el papel desempeñado por los "derivados de las letras", no puede negarse el impacto que estos medios producen en la sociedad. Por ello, antes de concluir el presente punto, se citarán resumidamente un conjunto de obras y artículos que, positiva o negativamente, ayudaron a acercar la robótica a la población.

### 2.2.3. Robots y Ficción: Mitología.

Para hablar del lugar ocupado por los robots en la ficción es necesario remontarse a relatos mitológicos y homéricos. Muchas mitologías antiguas tratan la idea de los humanos artificiales. En la mitología clásica, se dice que Cadmo sembró dientes de dragón que se convertían en soldados, y Galatea,

la estatua de Pigmalión, cobró vida. También el dios griego de los herreros, Hefesto (Vulcano para los romanos) creó sirvientes metálicos mecánicos inteligentes, otros hechos de oro e incluso mesas que se podían mover por sí mismas. Siendo, según la Ilíada, algunos de estos autómatas, los que ayudan al dios a forjar la armadura de Aquiles. Aunque, por supuesto, no se describe a esas máquinas como "robots" o como "androides", son en cualquier caso dispositivos mecánicos de apariencia humana.

También, es posible encontrar una leyenda hebrea que habla del Golem, una estatua animada por la magia cabalística. Por su parte, las leyendas de los Inuit describen al Tupilaq (o Tupilak), que un mago puede crear para cazar y asesinar a un enemigo. Sin embargo, emplear un Tupilaq para este fin puede ser una espada de doble filo, ya que la víctima puede detener el ataque del Tupilaq y reprogramarlo con magia para que busque y destruya a su creador.

Aunque alejados de la idea actual de robot, estos ejemplos son muestra evidente de las ansías humanas por contar con "máquinas" o artilugios que realicen trabajos por sí mismos. Mediante estas citas se justifica que la idea de robots y autómatas no surge, en la sociedad, como resultado de la innovación tecnológica, sino que, en la mente humana, siempre ha existido la idea de ingeniar máquinas que le libren de sus quehaceres, o en su defecto los faciliten.

### 2.2.4. Robots y Ficción: Literatura.

Son muchas las obras en las que se hace referencia a robots. Sin ir más lejos, en el conocido cuento de "El Mago de OZ", uno de los protagonistas es un robot. Ya en 1817, en un cuento de Hoffmann llamado *El hombre de arena*, aparece una mujer que parecía una muñeca mecánica, y en la obra de Edward S. Ellis de 1865, *El Hombre de Vapor de las Praderas,* se expresa la fascinación americana por la industrialización.

Como se indicaba anteriormente, la primera obra en utilizar la palabra *robot* fue la obra teatral R.U.R. de Čapek, (escrita en colaboración con su hermano Josef en 1920; representada por primera vez en 1921; escenificada en Nueva York en 1922. La edición en inglés se publicó en 1923). La obra comienza

en una fábrica que construye personas artificiales llamadas robots, no obstante, éstas están más cerca del concepto moderno de androide o clon, en el sentido de que se trata de criaturas que pueden confundirse con humanos.

En cuanto a la acuñación de la palabra robótica, según el Oxford English Dictionary, el principio del relato breve *¡Mentiroso!* de 1941 contiene el primer uso registrado de la palabra robótica. El autor no fue consciente de esto en un principio, y asumió que la palabra ya existía por su analogía con mecánica, hidráulica y otros términos similares que se refieren a ramas aplicadas del conocimiento.

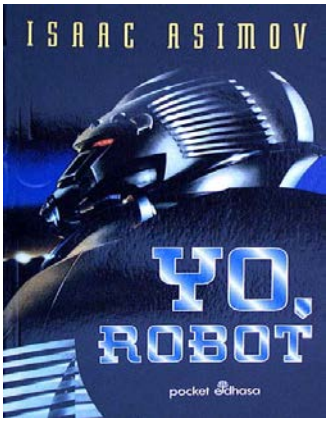



*Ilustración 5.Portada del libro "Yo, Robot" de I.Asimov.*

Sin embargo, el autor más prolífico de historias sobre robots fue, sin lugar a dudas, Isaac Asimov (1920-1992), que colocó los robots y su interacción con la sociedad en el centro temático de muchos de sus libros. Tal es el grado de descripción e invención alcanzado por las obras de Asimov que muchos lo consideran un visionario o "Verne de la Robótica". Este autor consideró cuidadosamente la serie ideal de instrucciones que debería darse a los robots para reducir el peligro que éstos representaban para los humanos, una vez alcanzado el punto de desarrollo en el que los robots pueden ser considerados "seres inteligentes". Así, llegó a formular sus famosas Tres Leyes de la Robótica:

- Ningún robot causará daño a un ser humano o permitirá, con su inacción, que un ser humano sufra daño.
- Todo robot obedecerá las órdenes que le den los seres humanos, a menos que esas órdenes entren en conflicto con la primera ley.
- Todo robot debe proteger su propia existencia, siempre que esa protección no entre en conflicto con la primera o la segunda ley.

Esas tres leyes se introdujeron por primera vez en su relato corto, de 1942, *Círculo Vicioso*, aunque habían sido esbozadas en algunos textos anteriores. Más tarde, Asimov añadió la ley de Cero: "Ningún robot causará daño a la humanidad ni permitirá, con su inacción que la humanidad sufra daño". Por lo que el resto de las leyes se modificaron para ajustarse a este añadido.

Llegados a este punto, conviene señalar que el impacto de la literatura de Asimov fue tal que, en el año 2011, el Consejo de Investigación de Ingeniería y Ciencias Físicas (Engineering and Physical Sciences Research Council, EPSRC por sus siglas en inglés) y el Consejo de Investigación de Artes y Humanidades (Arts and Humanities Research Council, AHRC por sus siglas en inglés) de Gran Bretaña publicaron conjuntamente un conjunto de cinco principios éticos "para los diseñadores, constructores y los usuarios de los robots en el mundo real, junto con siete mensajes de alto nivel", destinado a ser transmitido, sobre la base de un taller de investigación en septiembre de 2010.

### 2.2.5. Robots y Ficción: Cine y Televisión.

El tono económico y filosófico iniciado por R.U.R. sería desarrollado más tarde por la película *Metrópolis*, y las populares *Blade Runner* (1982) o *The Terminator*(1984). Esta última ha calado muy hondo en la mentalidad de la sociedad actual, especialmente de las generaciones de los 80 y los 90. Gran parte de la población piensa que el desarrollo de los drones, la inteligencia artificial y las redes neuronales, desencadenarán, a la larga, una situación similar a la protagonizada por Arnold Schwarzenegger. Así, en el campo de los robots, de los humanoides y los drones, la robótica no sólo se enfrenta a las controversias y polémicas inherentes a aspectos políticos como puede ser la normativa de vuelo para RPAS (Remotely Piloted Aircraft Systems) sino que también debe afrontar barreras y miedos sociales. Estas generaciones tampoco olvidarán el variopinto elenco de robots y drones que forman parte de la trilogía de la Guerra de las Galaxias. Si bien esta saga no plantea ni abre ningún tipo de debate si muestra una gran variedad de robots: móviles, antropomórficos,

*Ilustración 5. Conocidos Robots de la Saga Star Wars*

manipuladores, escaladores… a la par que muestra cómo estos se han integrado en la sociedad e interactúan con ella.

Existen muchas otras películas sobre robots, entre ellas cabe destacar: *Inteligencia Artificial (IA)*, un film mediante el que se plantea el debate sobre el grado de similitud entre los seres humanos y los robots. También deben citarse las dos películas basadas en los relatos de Isaac Asimov: *Yo, Robot* y *El hombre bicentenario*, en las que se abordan diversos temas como pueden ser la robótica aplicada a la medicina, el desarrollo de implantes robóticos y las consecuencias de desarrollar redes neuronales al extremo.

En televisión, existen series muy populares como Robot Wars, *BattleBots y* una serie de la taquillera Saga de R2D2 ya citada. Por otro lado, en la serie *Futurama* de Matt Groening, los robots poseen una identidad propia, como ciudadanos, así como se encuentran dotados de una actitud y consciencia propia que les caracteriza. Tampoco olvidar, la conocida serie RoboCop que expone la temática de las prótesis robóticas y las ventajas que están pueden ofrecer para complementar, sustituir y/o mejorar las propias funciones y capacidades humanas. Por último, en la serie *Almost Human,* aparecen robots-policías con conciencia propia, llamados DRN, los cuales funcionan con un programa de "alma sintética".

Tras estas breves referencias se manifiesta la importancia que ha tenido la ficción en el acercamiento de los robots a la sociedad. Se han mostrado además ejemplos que simplemente tratan de crear polémica y abrir debates con objeto de concienciar a la masa y/o informarla. Por otro lado, también han sido comentados casos en los que la séptima maravilla más que ayudar no ha hecho sino entorpecer el camino al abrir nuevos horizontes de disputa que deben ser solventados antes de aprobar leyes relativas al desarrollo de la robótica y obligaciones de sus promotores. Concluir señalando que a pesar de ello, con el transcurso de los años, la robótica va haciéndose hueco en la sociedad, eliminando tabúes y miedos a base de mostrar sus posibilidades y los beneficios que puede reportar con su desarrollo.

## 2.3. Contexto actual del proyecto.

En cuanto a hardware, el proyecto puede quedar divido conceptualmente en dos términos de gran extensión. Por un lado se encuentra el núcleo del sistema, robot a controlar, y por otro los "sensores", ayudas y complementos al robot; para este caso, un sistema de visión artificial mediante el que reconocer el objeto de agarre.

La pieza angular en torno a la que gira el conjunto del presente proyecto es un robot manipulador antropomórfico denominado Shadow Hand. Una imagen del mismo puede observarse a continuación. Aunque la descripción del conjunto de habilidades y características del robot se desarrollará a lo largo de futuros capítulos, se realizará una breve descripción del mismo con objeto de facilitar su contextualización y justificar el posicionamiento realizado, dentro del paradigma definido por la robótica contemporánea.

Sin más preámbulos, la mano robótica Shadow Hand queda englobada dentro de la subclase de los robots manipuladores. De su complexión, articulaciones y posibilidades de agarre se extiende que no se trata de un mero robot manipulador de los que suelen emplearse más comúnmente en industria, sino que, por el contrario, queda descrito como una evolución o mejora de estos sistemas. Con el devenir de los tiempos y el ferviente desarrollo tecnológico logrado por la comunidad científica e ingenieril de la época, los costes de fabricación de accionamientos, electrónica y diverso material mecánico de tamaño micro ha sido reducido drásticamente. Este hecho unido al grado de calidad y prestaciones alcanzadas, en el desarrollo de piezas, ha posibilitado la producción de nuevos sistemas entre los que figuran este grupo de manipuladores "modernos" o de primera clase.

*Ilustración 6. Mano robótica Shadow Hand con 24 GDL.*

Como características relevantes del robot convienen destacar sus 24 grados de libertad, el juego de muñeca integrado y las posibilidades de configuración. Sin entrar en detalles específicos y funcionales adicionales, el manipulador se encuentra disponible en dos modelos: modelo *Ethercat*, construido con accionamientos eléctricos, y el modelo *Air- Muscle* que destaca por su accionamiento neumático, permitiendo agarrar objetos de elevada fragilidad.

A continuación, se enumerarán un conjunto de robots manipuladores presentes en el mercado actualmente. Antes de tratar a comparar unos con otros, el lector debe tener en cuenta que la mano robótica Shadow Hand fue comercializada en el año 2002, alcanzando el título de pionera de los robots manipuladores de "alta gama".

### 2.3.1. Barrett Hand.


*Ilustración 6.Barrett Hand*

La serie BH8 de Barrett corresponde a una pinza multidedo programable con la capacidad de asegurar el agarre de objetos de diferentes tamaños, formas y orientaciones. Pese a su reducido peso (980 gramos) y forma compacta, está completamente integrada y posee plena funcionalidad. Su comunicación se encuentra basada en comunicaciones serie estándar, destacando la rápida y sencilla interfaz de programa que permite configurarla fácilmente para trabajar con cualquier brazo robótico del mercado. Gracias a ello, según la empresa:

> *"la serie BH8 multiplica inmediatamente el valor de cualquier brazo que requiera automatización flexible".*

En cuanto a hardware, la Barrett Hand integra una CPI, software, electrónica de comunicación, servo-controladores y 4 motores sin escobillas (brushless). De sus tres dedos de articulaciones múltiples, dos están dotados de un grado extra de movilidad que permite su giro en torno de la base.

### 2.3.2. Mano SVH de Schunk.

La SVH de Schunk es una mano robótica avanzada, que trata de reproducir lo más fielmente posible los 27 grados de libertad, el tamaño, la forma, el aspecto y la movilidad de una mano humana.

*Ilustración 7.Mano robótica SVH.*

Para lograr tamaña hazaña, se han dispuesto nueve drivers que permiten a la mano, de cinco dedos, llevar a cabo diversas operaciones de agarre. Tal es el grado de reproducibilidad alcanzado que, gran parte de los gestos realizados por una mano humana real pueden ser replicados. De este modo, no sólo se logra simplificar la comunicación visual entre un robot humano y el servicio, sino que, además, se incrementa el nivel de aceptación en el entorno doméstico.

Para facilitar el agarre se han dispuesto una serie de sensores táctiles en los dedos, así el robot hace gala de la sensibilidad necesaria para agarrar un variado conjunto de objetos de distintas características geométricas, volumétricas, materiales, etc., pudiendo así gestionar todas las tareas de *grasping* y de manipulación planteadas, incluso en situaciones no estructuradas e imprevisibles. Por último, destacar la superficie elástica de agarre de que consta y mediante la cual se garantiza una sujeción fiable de los objetos agarrados.

La mano SVH está completamente integrada, localizando el núcleo de comunicaciones en la muñeca, pudiéndose conectar con los brazos ligeros a través de interfaces definidas.

### 2.3.3. EH1 Milano Hand.

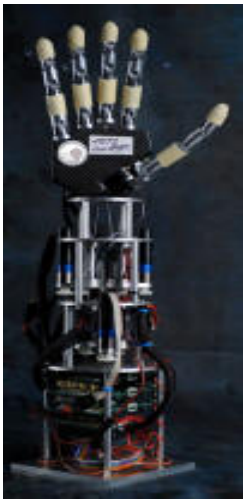


*Ilustración 8.EH1 Milano Hand.*

La mano EH1 Milano de Prensilia S.L es una mano antropomórfica programable de tamaño humano que permite agarrar un juego variado de objetos mediante el uso de diversos sensores de fuerza y posición. Una de las bazas de esta mano se encuentra en la comunicación por cable de que dispone y su reducido peso, en torno a 250 gramos, lo que permite que pueda ser posicionada por brazos robóticos con bajos valores de carga máxima. En cuanto al hardware, señalar que cada actuador contiene; una CPU, el firmware necesario para trabajar con él, la electrónica necesaria para interactuar con sensores y actuadores, los dispositivos de comunicación inherentes, servo-controladores, y un motor de corriente continua con escobillas. La mano se comunica a través de protocolo RS232 o USB, y ha sido configurada de forma tal que permita ser fácilmente integrada en un variopinto conjunto de escenarios de investigación que van desde el uso de prótesis, la neurociencia, la interacción humano-robot, rehabilitación, etc.

### 2.3.4. ELU-2 Hand.

*Ilustración 9. Elu-2 Hand.*

El robot Elu-2 Hand desarrollado por la empresa Elumotion.Ltd, puede describirse como un manipulador antropomórfico con el aspecto y dimensiones de una mano humana. Esta mano posee tan sólo 9 grados de libertad, conseguidos mediante el uso de servomotores accionados eléctricamente. Al igual que otros modelos de manos ya presentados, su diseño tuvo en cuenta la compatibilidad con brazos robóticos de otras marcas. Esta mano se define como manipulador dotado de sensores de tacto y una configuración software tal que permita su operación en aplicaciones críticas.

### 2.3.5. MLR Hand.

*Ilustración 10. MLR Hand.*

Se trata de una prótesis robótica desarrollada como parte de un programa del Laboratorio de Física Aplicada Stephen Hopkins. Este biorobot fue utilizado por los soldados heridos del Centro Medico Militar Nacional Walter Reed, en el año 2012. La parte superior del brazo contiene un hombro con dos motores que permiten los movimientos de abducción / aducción y flexión / extensión, un rotador del húmero, un codo, y la unidad de potencia. Dependiendo del nivel de amputación del paciente receptor en particular, se realizan diversas variaciones del conjunto de articulaciones y mecanismos a instalar. Los movimientos de los dedos se diseñaron como unidades independientes con controladores de motores internos que permiten controlar la posición. Así mismo, cada dedo contaba con la electrónica necesaria para establecer la comunicación con la palma.

### 2.3.6. i-Limb Hand.

*Ilustración 12. i-Limb Hand*

Se trata de una prótesis robótica similar a la anterior desarrollada por Touch Bionics, UK. Esta prótesis consta de un sistema de control muy sencillo e intuitivo basado en la adquisición de los impulsos enviados por los músculos anexos al tratar el paciente de mover la prótesis como si fuese una parte natural de su cuerpo. Para ello se adquieren, mediante electrodos colocados sobre la superficie de la piel, las señales eléctricas generadas por los músculos en la parte extrema sobre la que se aplica la prótesis. Al igual que otros modelos similares, los dedos se encuentran alimentados individualmente con objeto de facilitar su mantenimiento y reparación, evitando privar al paciente del conjunto funcional.

### 2.3.7. Bebionic 3.

*Ilustración 11. Usuario de la prótesis Bebionic.*

En otra línea se encuentra la prótesis robótica desarrollada por *Bebionic*. Esta prótesis destaca por la potencia de sus motores y su accionamiento basado en impulsos nerviosos. Además ofrece posibilidad de realizar 14 sujeciones distintas, desde la sujeción de apretón que puede ser utilizada para agarrar un balón y lanzarlo, hasta la sujeción del ratón de un ordenador.

La mano también se puede configurar y adaptar a las necesidades del cliente según los requisitos individuales del usuario. Para esta finalidad, la empresa ha desarrollado una herramienta software, *beBalance*, que permite que la mano sea configurada por un médico.

### 2.3.8. Open Hand Project.

Aunque esta mano robótica no puede equipararse a los modelos comerciales presentados anteriormente, el matiz e intenciones de la iniciativa justifican su aparición en este listado de manipuladores robóticos. La idea fundamental de este proyecto es diseñar manos protésicas avanzadas de bajo coste. Obviamente, esta limitación económica impone, de forma inherente e imperativa, un conjunto de barreras que imposibilitan obtener las funcionalidades y prestaciones de que gozan los modelos anteriormente presentados.

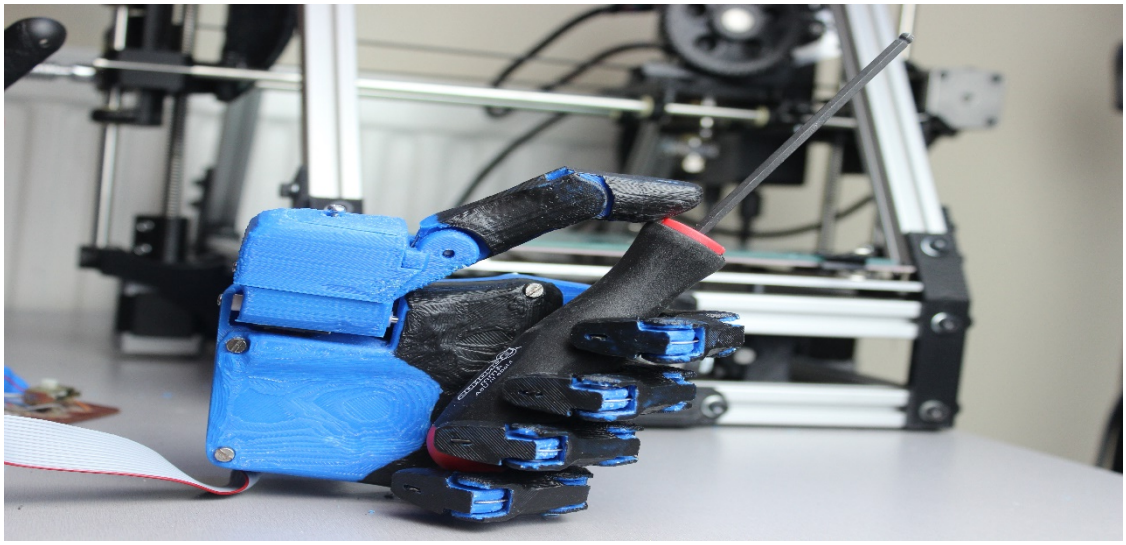


*Ilustración 13. Prototipo de la iniciativa Open Project Hand*

A día de hoy, la fase de desarrollo del proyecto se encuentra situada en la mejora de un prototipo recientemente desarrollado. Aunque escasean los detalles, que pueden ser consultados en la página web del proyecto junto a las bases sociales y económicas (https://www.indiegogo.com/projects/the-open-hand-project-a-low-cost-robotic-hand), se pueden extraer una serie de datos que permiten tener una idea del funcionamiento y arquitectura de la mano.

Según se detalla, la mano desarrollada se encuentra accionada por motores eléctricos, para la estructura se han empleado impresiones realizadas en plástico mediante el uso de impresores 3D. Además, señalan que se ha dispuesto la electrónica necesaria para su control, aunque sin profundizar en más detalles. En cuanto a la alimentación y sincronización de los motores, se sabe que éstos se alimentan de forma individual y el sistema dispone de medios para detectar cuando el movimiento de un dedo se encuentra obstruido. Como se desprende de esta definición, el proyecto no trata de competir con las prótesis robóticas

existentes hoy día sino que su objetivo se encauza hacia facilitar la adquisición de prótesis sencillas que mejoren ligeramente la calidad de vida de las personas, principalmente de aquellas con limitados recursos económicos que no pueden costearse los gastos que supone la adquisición de una prótesis más avanzada.

# CAPÍTULO 3
# SOFTWARE APLICADO A LA ROBÓTICA.

## 3.1. Introducción

El marco de crecimiento presentado por la robótica, rápido y en ocasiones, descoordinado, constituyó un lastre para el continuo desarrollo de la disciplina. La aparición, cada vez con mayor frecuencia, de nuevos robots basados en pautas de comportamiento de disparidad creciente, ocasionó la eclosión de múltiples medios y lenguajes privativos de programación. De este modo, se comprueba cómo la empresa Shadow, por ejemplo, proporcionaba sus propias API's y software de manejo. Por otro lado, podemos encontrar a los robots KUKA con una filosofía de programación y "buen hacer de las cosas" totalmente distinta.

Esta aglomeración de posibilidades de programación específicas, unida a la dificultad intrínseca a la materia en cuestión, no suponen sino un *hándicap* a cumplir por los profesionales. Viéndose éstos abocados a la especialización en un cierto de tipo de robot muy específico y limitándose tanto las opciones de expansión como la colaboración entre los desarrolladores.

Así, a lo largo del presente capítulo, se expondrán las principales tendencias actuales, en cuanto a software robótico, que tratan de favorecer la colaboración entre desarrolladores y facilitar la programación en la medida de lo posible.

## 3.2. Sistemas de Tiempo Real.

De forma general, se puede enunciar que un sistema robótico está constituido por un amplio conjunto de actuadores y sensores cuya relación queda establecida por el entramado lógico implementado en él. Además, si consideramos la forma en que se relaciona dicho sistema con su entorno comprobamos que unívocamente queda definido como sistema de tiempo real.

Según la literatura, un sistema en tiempo real puede definirse como todo sistema informático que:

- Interacciona repetidamente con su entorno físico.
- Responde a los estímulos que recibe del mismo dentro de un plazo de tiempo determinado.
- Impone exigencias de:
  - Concurrencia.
  - Fiabilidad y Seguridad.
  - Determinismo temporal.

De dicha definición puede extraerse que el correcto funcionamiento de un sistema en tiempo real no sólo exige que las acciones realizadas sean correctas, sino que añade una necesidad temporal de actuación a cumplir.

De esta definición se deduce que; uno de los aspectos que más se han de cuidar, a la hora de desarrollar software, es el sistema operativo seleccionado para ello. Éste impondrá una serie de limitaciones y determinará la forma en la que el software se relaciona con el hardware.

Como resultado de analizar las características de un sistema de tiempo real se obtiene que los sistemas operativos convencionales resultan inadecuados para aplicaciones de tiempo real ya que:

- No presentan comportamiento determinista.
- Se muestran incapaces de garantizar los tiempos de respuesta.
- Gozan de dudosa fiabilidad.

A estas objeciones se deben sumar los atributos esperados de un sistema operativo de tiempo real:

- Concurrencia: Ejecución de procesos ligeros (threads) con memoria compartida.
- Temporización.
- Planificación determinista: prioridades fijas con desalojo.
- Acceso a recursos hardware e interrupciones.

En segundo lugar, no debemos olvidar el lenguaje de programación escogido para llevar a cabo la ardua tarea de programación. Al igual que ocurre con los sistemas operativos, no todos los lenguajes de programación son aptos para la programación en tiempo real. De forma resumida, la adecuada o incorrecta selección del lenguaje de programación se traducirá en términos de fiabilidad y eficiencia del desarrollo. Por ello, es imprescindible seleccionar un lenguaje que proporcione las primitivas de diseño ajustadas a los requisitos del sistema.

## 3.3. Linux como núcleo del sistema.

En la actualidad pueden encontrarse gran cantidad de sistemas operativos, cada uno de ellos con características totalmente distintas. En términos generales, una primera división se realizaría entre sistemas basados en MS-DOS (Windows) y aquellos basados en arquitectura UNIX, entre los que destacan Macintosh OS y Linux (con todas sus distribuciones). En la siguiente tabla, se puede observar una sencilla comparativa entre estos tres sistemas operativos,

(Tabla realizada a partir de la información referida en [1] [2].)

*Tabla 1. Principales Sistemas Operativos.*

| | Mac OS  | Windows   | Linux  |
|---|---|---|---|
| Distribución Actual. | Mavericks y Mountain Lion | Windows 8 y Windows 8.1 | Específico de cada distribución. |
| Procesadores válidos. | Intel (diseño propio del hardware) | Intel y AMD | Intel y AMD |

1 http://wwwjosejuansantaella.blogspot.com.es/2012/10/tabla-comparativa.html.

2http://1.bp.blogspot.com/_TOh1rMOoRvI/R5jMn0kzMRI/AAAAAAAAAEg/WNzs6C3J3oo/s1600/comparativa+SO.JPG

| | | | |
|---|---|---|---|
| Coste de Licencia. | En torno a los 100-150 euros. Mavericks sólo requiere poseer un ordenador de la marca. | En torno a los 120 euros. | Gratuito, salvo contadas excepciones. |
| Complejidad de Manejo. | Baja | Baja | Alta |
| Principales Características. | 1. Aspecto gráfico y facilidad de uso muy cuidados.<br>2. Programación modular y mayor estabilidad que otros sistemas.<br>3. Las memorias son DIMM de 168 contactos, y los discos de tecnología SCSI II.<br>4. Dispositivos de entrada, por defecto, inalámbricos.<br>5. Uso de hardware específico.<br>6. Puerto Thunderbolt de comunicaciones.<br>7. Multiusuario.<br>8. Protección de memoria entre procesos. | 1. Interfaz gráfica con menús desplegables, ventanas en cascada y soporte para mouse.<br>2. Gráficos de pantalla e impresora independientes del dispositivo.<br>3. Multitarea cooperativa entre las aplicaciones Windows.<br>4. Ventanas traslapadas.<br>5. Archivos PIF para aplicaciones DOS.<br>7. Modo estándar (286), con soporte de memoria grande (largo, menor).<br>8. Multiusuario. | 1. Multitarea: varios programas (realmente procesos) ejecutándose al mismo tiempo.<br>2. Multiusuario: varios usuarios en la misma máquina al mismo tiempo.<br>3. Multiplataforma: corre en muchas CPU distintas, no sólo Intel.<br>4. Funciona en modo protegido 386.<br>5. Tiene protección de la memoria entre procesos, de manera que uno de ellos no pueda colgar el sistema.<br>6. Interfaz gráfica opcional. |
| Ventajas. | • Mejor interfaz gráfica del mercado.<br>• Ideal para diseño gráfico.<br>• Gran estabilidad. | • Más conocido y usado.<br>• Gran cantidad de ayuda.<br>• Fácil de usar. | • Gran cantidad de software libre para este sistema.<br>• Mayor estabilidad. |

| | | | |
|---|---|---|---|
| | | • Más software desarrollado. | • Existen distribuciones de Linux para diversos tipos de equipo, incluso versiones light o adaptadas a grandes limitaciones hardware.<br>• Las vulnerabilidades son detectadas y corregidas más rápidamente que cualquier otro sistema operativo.<br>• Mayor tasa de actualizaciones.<br>• Gran comunidad de usuarios.<br>• Gran comunidad de usuarios.<br>• Corrección de código es más rápida, una vez se domina el uso del terminal. |
| Inconvenientes. | • Costoso (aunque viene incluido con la maquina)<br>• Existe poco software para este sistema operativo.<br>• Es más complicado encontrar gente que la pueda arreglar en caso de fallos.<br>• Grandes inconvenientes a la hora de establecer comunicaciones con el exterior. | • El costo es muy alto<br>• Las nuevas versiones requieren muchos recursos<br>• La mayoría de los virus están hechos para Windows.<br>• Puedes tener errores de compatibilidad en sistemas nuevos.<br>• Históricamente es el más inestable de los 3 sistemas. | • Necesita usar la línea de comandos.<br>• Gran parte de los programas no están desarrollados para esta plataforma.<br>• Cualquier tipo de instalación requiere de un proceso de mayor complejidad. |

| | | | |
|---|---|---|---|
| | • Sistema poco flexible en cuanto a configuración.<br>• Precio de los programas. | | |

Aunque no se menciona en la tabla, ninguno de los tres sistemas contempla la realización de procesos en tiempo real. No obstante, los gestores de procesos se encuentran tan optimizados que dan la sensación de incorporar este tipo de ejecución. Este hecho, que puede resultar irrelevante a la hora de realizar programas convencionales, cobra especial importancia a la hora de gobernar sistemas que, por definición propia, precisan interactuar de forma continua con su entorno.

Es en este momento cuando las ventajas que Linux ofrece muestran todo su potencial. Sin tratar de entrar en detalle y evitando realizar una definición puramente informática, el sistema de Linux puede definirse "grosso modo" como un núcleo denominado Kernel responsable del comportamiento básico del sistema: gestión de procesos, aplicaciones, control del hardware... Sin embargo, este núcleo puede ser mejorado mediante la adición de módulos, lo que se realiza activando paquetes de código. En otras palabras, se tiene un sistema básico cuyas características pueden mejorarse mediante la compilación de extensiones.

De este modo, entre los múltiples módulos compilables del Kernel de Linux se encuentra uno en cuya denominación incorpora las siglas RT (el nombre completo puede variar ligeramente dependiendo de la distribución) que dota al sistema de capacidades de tiempo real. Es este módulo el que permite que usemos las distribuciones típicas de Linux para trabajar con sistemas de tiempo real en lugar de tener que recurrir a sistemas específicos que satisfagan estas necesidades temporales.

El segundo motivo por el que se ha escogido el sistema operativo Linux es, como se puede deducir de lo antes expuesto, el potencial oculto que presenta el sistema: sus capacidades y funcionalidad pueden incrementarse hasta límites insospechados mediante la activación de diversos módulos que complementen al sistema principal, siempre y cuando el hardware lo permita.

Por último, y no por ello menos importante, es necesario destacar uno de los puntos más fuertes de Linux; la seguridad. En Linux, los virus son una amenaza menor. Aparte de la escasez de virus diseñados para atacar al sistema, para realizar cualquier configuración o modificación importante sobre el sistema (lo que puede ser crear una carpeta en el fichero de archivos) es necesario gozar de permisos de *superusuario* (denominado administrador en otros sistemas), algo que no se consigue tan fácilmente. Además, Linux permite realizar gran cantidad de pruebas y análisis, entre ellos se encuentran las pruebas de seguridad de la red. Y es que Linux es un gran instrumento para encontrar vulnerabilidades en el sistema que podrían aprovechar los piratas informáticos (hackers) para robar información. Además, es posible realizar acciones de seguridad tales como cifrar todo el disco duro o ser *invisible* en la red de Internet.

A pesar de todas estas ventajas, Linux posee un arma de doble filo resumida en una frase característica de la comunidad de usuarios del sistema: "*Linux es un Sistema Operativo de informáticos para informáticos*". Esta frase agrupa toda la esencia del sistema y permite explicar gran parte de las decisiones tomadas a la hora de desarrollarlo. A diferencia de otros sistemas operativos, el objetivo primordial de Linux no es ser intuitivo o vistoso, sino que su principal baza reside en la potencia descontrolada que proporciona al usuario, con escasas limitaciones o seguridades en este aspecto. Al dotar al usuario de tal poder sobre el sistema, éste adquiere una capacidad de control y gobierno que, aunque beneficiosa, puede causar la corrupción de todos los ficheros que lo integran. Este hecho queda claramente evidenciado con los primeros pasos que demos en Linux y es que lo más frecuente es reinstalar Linux tras el primer mes de uso, precisamente por la gran libertad para hacer y deshacer que otorga.

### 3.3.1. Distribuciones de Linux.

Una vez escogido el sistema operativo, Linux, queda lidiar con qué distribución escoger. A diferencia de otros sistemas, Linux, al ser open-source, ha sido desarrollado en distintas versiones, cada una de ellas con sus propias características y particularidades, denominadas distribuciones. Una definición más formal de distribución sería la que da el portal oficial de Linux España:

*"Una distribución no es otra cosa, que una recopilación de programas y ficheros, organizados y preparados para su instalación. Estas distribuciones se pueden obtener a través de Internet, o comprando los CDs de las mismas, los cuales contendrán todo lo necesario para instalar un sistema Linux bastante completo y en la mayoría de los casos un programa de instalación que nos ayudara en la tarea de una primera instalación."* [3][3]

Una vez definido el concepto queda valorar las distintas alternativas que se presentan para trabajar con un sistema de tiempo real. En la siguiente tabla se puede observar un desglose de características de las distribuciones de Linux más extendidas a fecha de Junio de 2014.

Toda la información de la tabla ha sido obtenida de sitios webs. [3]

*Tabla 2. Principales Distribuciones de Linux.*

| Distribución | Breve Descripción/ Características. |
|---|---|
| ubuntu | Distribución basada en Debian, con lo que esto conlleva, y centrada en el usuario final y facilidad de uso. Muy popular y con mucho soporte en la comunidad. El entorno de escritorio por defecto es GNOME. |
| redhat. | Esta es una distribución que tiene muy buena calidad, contenidos y soporte a los usuarios por parte de la empresa que la distribuye. Es necesario el pago de una licencia de soporte. Enfocada a empresas. |
| fedora | Esta es una distribución patrocinada por RedHat y soportada por la comunidad. Fácil de instalar y buena calidad. |
| debian | Otra distribución con muy buena calidad. El proceso de instalación es quizás un poco más complicado, pero sin mayores problemas. Gran estabilidad antes que últimos avances. |
| openSUSE | Otra de las grandes. Fácil de instalar. Versión libre de la distribución comercial SuSE. |

3 WWW.LINUX-ES.ORG/DISTRIBUCIONES

| | |
|---|---|
|  | Distribución basada en Ubuntu, con lo que esto conlleva y centrada en el usuario final y facilidad de uso. La gran diferencia con Ubuntu es que el entorno de escritorio por defecto es KDE. |
|  | Esta distribución fue creada en 1998 con el objetivo de acercar el uso de Linux a todos los usuarios, en un principio se llamó Mandrake Linux. Facilidad de uso para todos los usuarios. |

Además de éstas, existen más tipos de distribuciones, cada una con sus propias características y orientada a un tipo específico de usuario. Uno de los aspectos que más llaman la atención y pueden confundir a la hora de escoger una distribución es el escritorio y las derivaciones. Al hablar de derivaciones, nos referimos a la procedencia o en qué distribución se basa cierta distribución, valga la redundancia. El primer cambio de mentalidad que debemos aplicar al sumergirnos en el mundo de Linux es la del concepto de open-source, la gran mayoría de los productos basados en Linux son gratuitos, por ello es frecuente encontrar versiones que derivan de otra más compacta con el simple objetivo de poder ser usadas en equipos de menor potencia, un ejemplo sería Kubuntu o Linux Mint. Así mismo, puede darse el caso contrario, en el que cierta distribución supone una mejora o incremento de potencia respecto a su "progenitora". Por otro lado, un aspecto banal que adquiere, a veces, una importancia inmerecida es el escritorio. Dada la variedad de distribuciones y gustos de los usuarios, existen distintos tipos de "escritorio", lo que se puede traducir en formas de representación de la información o entorno gráfico. En caso de no ser muy escrupuloso, lo usual es escoger la distribución de Linux que mejor satisfaga nuestras necesidades en cuanto a rendimiento y maniobrabilidad, quedando relevados a un segundo plano los aspectos gráficos.

Aunque se podría proseguir con este tema durante innumerables páginas, se ha considerado que por limitaciones de tiempo, espacio y claridad, este apartado pretende ser simplemente una familiarización o introducción con el entorno de Linux. Su instalación, configuración y manejo será abordada en profundidad durante los capítulos relativos a la configuración del software.

## 3.4. Middleware: La segunda parte del núcleo.

Los sistemas robóticos requieren de una interacción y coordinación entre diversos elementos software y hardware caracterizada especialmente por una elevada complejidad. De este modo, los diseños de las distintas arquitecturas software y hardware deben ser realizadas con sumo cuidado, manteniendo siempre en mente la unión entre ambas partes que deberá acaecer. Resulta obvio, que esta integración adquiere creciente complejidad e importancia conforme el número de robots y tipos de ellos integrados en el sistema a controlar incrementa.

Con el desarrollo tecnológico, un término relativo a la ingeniería de software comenzó a cobrar especial importancia, pasando a ser un invitado común en toda reunión relacionada con la temática: middleware robótico. Dado el crecimiento experimentado por la robótica y las ansias expansionistas existentes, el conjunto de la comunidad de especialistas coincide en la necesidad de adoptar un sistema de desarrollo que establezca un marco común que permita compartir conocimientos, diseños y reciclar código.

La palabra middleware fue acuñada por primera vez en 1968 cuando d'Agapeyeff la usó para referirse a aquellas partes de software que puede ser compartidas por distintas aplicaciones. Con el transcurso del tiempo, el término ha ido ganando definiciones. Según señalan [D.Gill y D.Smart] [4] citando a [Bakken.et al] [5], al hablar de middleware, hacemos referencia a:

"Una clase de la tecnología de software diseñada para ayudar a manejar la complejidad y heterogeneidad inherente a los sistemas distribuidos. Pudiendo ser considerada una de las capas inferiores de software, situada entre la capa de sistema operativo y la de aplicación, que proporciona un paradigma común de abstracción en cuanto a programación."

4 [D.Gill y D. Smart] Middleware for Robots? Christopher D. Gill and William D. Smart, Washington University in St. Louis

5 [Bakken at.el] Bakken, D. 2001. Middleware. In Urban, J., and Dasgupta,P., eds., Encyclopedia of Distributed Computing. Kluwer.to appear.

Migliavacca [6] complementa esta definición al señalar que, el middleware puede ser visto como un conjunto de servicios que permiten ejecutar múltiples procesos simultáneamente, los cuales interactúan, a su vez, con una o más máquinas. Este mismo autor, enumera una serie de características que justifican la importancia y aceptación adquirida por los middleware durante los años. Según el autor, los middleware se caracterizan por ser:

- Portables. Ofrecen un modelo de programación, independiente del lenguaje escogido y sistema operativo empleado. Este hecho, supone una de las principales bazas de este tipo de software al facilitar enormemente el desarrollo y la reutilización, facilitando la tarea de los desarrolladores.

- Fiables. Este hecho queda justificado por el desarrollo y verificaciones a los que son sometidos antes de ser empleados en su aplicación final. Lo que permite la abstracción a aspectos de bajo nivel y promueve la utilización de librerías seguras y ampliamente verificadas.

- Permiten manejar su complejidad. Esta característica deriva de la anteriormente mencionada. La posibilidad de abstracción a bajo nivel proporciona dos grandes ventajas. Por un lado, reduce la probabilidad de errores en gran medida. Mientras que, por otro lado, simplifica enormemente el proceso de desarrollo.

6 [Migliavacca, Ceriani] Middleware in robotics. Simone Ceriani, Martino Migliavacca. (INTERNAL REPORT FOR "ADVANCED METHODS OF INFORMATION TECHNOLOGY FOR AUTONOMOUS ROBOTICS", PROF. G. GINI)

Aunque existe una amplia variedad de software de tipo middleware, cuya clasificación y análisis ha sido objeto de múltiples estudios. Dada la temática del presente trabajo, se abordará únicamente el análisis de las distintas opciones de middleware aplicado a la robótica.

Como es bien conocido, los sistemas robóticos se definen como sistemas complejos cuya eficiencia radica en la correcta comunicación y sincronización entre un amplio y variopinto conjunto de componentes hardware y software. En términos generales, un robot ejecuta un software responsable de coordinar todas las tareas a realizar. Así, el software debería disponer los tiempos de lectura de los sensores, la decodificación de los estímulos captados, el tratamiento de la información adquirida, la evaluación de las acciones de control a aplicar y, por último, la aplicación de las mismas. En un principio, el desarrollo de este tipo de software no trataba de adaptarse a ningún estándar, tampoco buscaba permitir su reutilización sino simplemente satisfacer y cumplir el propósito esencial para el que estaba siendo desarrollado: programar el comportamiento de un robot específico.

De lo expuesto, se evidencia, dadas sus características, el papel que puede desempeñar el middleware en este escenario. La aplicación de este tipo de software permite la mejora, organización, reutilización, mantenimiento y eficiencia del código desarrollado y por desarrollar en el campo. En otros términos, es posible señalar que, dicho software es el elemento que permite desarrollar modularmente un programa de forma independiente para unir posteriormente todos estos bloques desarrollados obteniéndose un programa de elevada potencia y funcionalidad.

No obstante, es en este punto donde surgen los inconvenientes. Vistas las posibilidades que ofrece el uso de middleware, fueron muchas las iniciativas que trataron de imponer su propio middleware para el campo de la robótica. Esto no hizo sino empeorar la situación ya que al ingente número de desarrollos diferentes se sumó la variedad de middlewares que proliferaron, cada uno con sus propios procedimientos y especificaciones. Entre los principales middlewares aplicados a robótica existentes conviene destacar los siguientes:

### 3.4.1. OROCOS

Este proyecto es considerado el promotor de la idea de proyecto de software libre para el control de robots. Nació en Diciembre del año 2000 de manos de EURON, motivado por dos décadas de experiencias fatídicas tratando de usar productos comerciales de software de robótica para investigación. En el momento de su aparición no existía otro tipo de software con estas características por lo que constituyó un proyecto de gran innovación e interés. De entre sus características destacó la licencia LGPL, su extrema modularidad y flexibilidad, su gran calidad, independiente, con librerías de tiempo real (RTT), basado en CORBA (Common Object Request Broker Architecture) y de cinemática y dinámica (KDL).

### 3.4.2. ORCA.

Es un entorno open-source orientado al desarrollo de sistemas basados en componentes robóticos. Para ello proporciona una serie de funciones y elementos que permiten generar bloques de código fácilmente embebibles en un sistema de mayor complejidad. Orca fue desarrollado a partir de OROCOS en el año 2003 por KTH Stockholm. Según sus desarrolladores, el punto clave era la reutilización de código. Para cumplir este objetivo se desarrollaron librerías específicas y aseguró el mantenimiento de los repositorios. Otro de los conceptos interesantes presentes en ORCA es la noción de proporcionar un medio para el desarrollo no restrictivo, con ello, sus creadores, deseaban facilitar la adaptación del software anterior a ORCA o desarrollado en otras plataformas, evitando reescribir código en la medida de lo posible. Una de sus principales diferencias entre Orca y OROCOS, según [MIGLIAVACCA, CERIANI][7] , reside en los toolkit o herramientas empleadas para construir el sistema de comunicaciones. En este caso, se comprueba que en Orca se sustituye CORBA, (usado desde 1991, también en OROCOS), por un framework más moderno: ZeroC: Internet Communication Engine (ICE).

### 3.4.3. ROS.

Aunque en un principio las siglas se referían a Robot Open Source, según señala [MIGLIAVACCA, CERIANI][7], con el transcurso de los años y la aceptación de que goza ROS actualmente, sus siglas cambiaron de nomenclatura, pasando a significar Robot Operating System. El desarrolló de ROS inició su andadura a finales de 2006 impulsado por el equipo de desarrolladores de Willow Garage, un laboratorio de investigación fundado en ese mismo año para acelerar el avance de la robótica open-source y los proyectos de carácter no-militar. ROS se definió como un sistema meta-operativo de código libre con expectativas de ser algo más que un simple middleware. Además de contemplar todos los objetivos de Orca y los middlewares éste provee una serie de funcionalidades que le equiparan a un sistema operativo: abstracción software, control de hardware a bajo nivel, envío de mensajes entre procesos… Además, ROS posee un sitio web, con gran partición de la comunidad, en el que los desarrolladores debaten y comparten conocimientos.

### 3.4.4. BRICS.

Se trata de un proyecto conjunto de investigación fundado por la Unión Europea que involucra a un gran número de participantes. La iniciativa BRICS busca identificar y documentar un conjunto de "buenas prácticas" que favorezcan el desarrollo de complejos sistemas robóticos, redefinir componentes ya existentes de forma que sea posible mejorarlos y reutilizarlos, y dar soporte al proceso de desarrollo mediante la disposición y creación de repositorios y herramientas de estructuración y organización.

### 3.4.5. YARP.

Sus siglas corresponden al acrónimo "Yet Another Robot Platform". Este software se define como un conjunto de librerías, protocolos y herramientas de código libre que permiten desarrollar módulos de programación, manteniendo los equipos desacoplados. Así mismo, en la definición que proporcionan los propios desarrolladores, se recalca que no debe confundirse con un sistema operativo ya que no implementa funciones que le asemejen, ni intenciones existen de ello. [YARP] [7] . Esta decisión se justifica a través de la intencionalidad multiplataforma y amigable que buscan sus desarrolladores, tratando de promover la reutilización de código y generando un entorno capaz de adaptarse fácilmente a sistema futuros. Para ello, YARP se desmenuza en tres partes esenciales, independientes del OS que lo soporte:

- **libYARP_OS**. Encargada de proporcionar una interfaz con el sistema operativo empleando procesamientos multi-hilo y *streaming* de datos. Esta librería se sirve de las funciones de código abierto ACE (ADAPTIVE Communication Environment) con objeto de ser neutra en cuanto a dependencia del sistema.
- **libYARP_sig**. Desarrolla las tareas de procesamiento común de señales (video y audio) y proporciona medios para comunicar con distintas opciones frecuentemente empleadas, como puede ser OpenCV.
- **libYARP_dev**. Proporciona una interfaz de comunicación con distintos equipos usados habitualmente en robótica: cámaras digitales, drivers de motores…

Aunque estos son los sistemas más relevantes o determinantes del camino recorrido por la robótica en el desarrollo de nuevas aplicaciones, existen una gran multitud de ellos. En la siguiente tabla obtenida de [Elkady y Sobh, 2011] [8], se puede observar un breve resumen y comparativa de todos ellos.

7 [YARP ] http://wiki.icub.org/yarpdoc/what_is_yarp.html

8 [Elkady y Sobh, 2011] Robotics Middleware: A Comprehensive Literature Survey and Attribute-Based Bibliography. Ayssam Elkady and Tarek Sobh. School of Engineering, University of Bridgeport, Bridgeport, CT 06604, USA. Received 21 August 2011; Revised 15 January 2012; Accepted 29 January 2012

## 3.5. Tabla Comparativa de middlewares para robótica.

| Name | System model | Control model | Fault tolerance | Simulator | Linux | Windows | Standards and technologies used | Open source | Behavior coordination | Real time | Distributed environment | Dynamic wiring | Security |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CLARAty | 2-layer AM; decentralized data; client server; platform independent abstractions and interfaces for various robotic components for motion control, coordination, mobility, manipulation, perception, estimation, navigation and planning. | Supports adaptation for centralized and distributed control; event driven | Yes | Yes | Yes | Only cygwin | OO design patterns, generic programming (C++ STL), ACE/TAO, CPPUnit, Qt, TCP, UDP, Doxygen | Partially | supported | Most modules are real time | Yes | Partially | Yes |
| Player | No particular architectural constraint, client-server, decentralized data | Not applicable; but on module level can be considered centralized, since it relies on polling model | No explicit fault handling capabilities | Stage is a 2D simulator, Gazebo is a 3D simulator | Yes | Yes | 3-Tier Architecture Proxy Objects | Yes | No | No | Yes | Yes | Yes |
| ORCA | No particular architectural constraint, peer to peer, decentralized data, component-based robotic systems | Not applicable; not clear on component level | No explicit fault handling capabilities | Yes | Yes | Yes | Ice | Yes | No | No | Yes | No | |
| MIRO | 3 layers, client-server, decentralized data | On object level there is an emphasis on event-driven control | No explicit fault handling capabilities | Yes | Yes | Yes | TAO Middleware C++ implementation of the CORBA standard | Yes | Yes | No | Yes | Yes | No |
| OpenRTM-aist | Component-based frame work, model-driven architecture, platform-independent model (PIM), Platform-specific model (PSM) | Component-based | Supported by RT-component model | OpenHRP3 is a dynamic simulator | Yes | Yes | CORBA | Yes | No | Yes | Yes | Yes | No |
| ASEBA | Event-driven distributed control | Event-based control | No | Yes | Yes | No | Event-based middleware, Virtual machines | Yes | Yes | Yes | Yes | Yes | No |
| MARIE | Component-oriented engineering approach, 3 layers | Centralized control unit | No | Yes | Yes | No | Mediator Interoperability technology, ACE | Yes | Yes | No | Yes | Yes | No |
| RSCA | Consists of real-time OS, communication middleware, and deployment middleware called core frame-work | Event-based control | No | No | Yes | Yes | POSIX. 13. CORBA. RT-CORBA vl. 1 | Yes | No | Yes | Yes | Yes | No |
| OPRoS | Component-based frame-works, validation/test tools | Client/server mechanism for control flow and the publisher/subcriber mechanism for data/event flow. | Being developed | Yes | Yes | Yes | event-driven | Yes | No | Being developed | Yes | Yes | No |
| ROS | Component-based frame-work, publisher/subscriber | Message oriented frame-works | Not Explicit | Yes | Yes | Partial functions | Message oriented, RPC services | Yes | Yes | Yes | Yes | Yes | No |
| MRDS | Component-based, REST | Distributed messaging | | Yes<br>No, but it | No | Yes | .NET/SOA | Comm. | Yes | No | Yes | Yes | Yes |

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| OROCOS | C++ libraries: OCL, KDL, BFL, | Event-based control | | No, but it has Orocos Simulik Toolbox | Yes | Yes | ACE/TAO, CORBA | Open source | No | Yes | No | Yes | No |
| SmartSoft | Service-oriented, component-based software, model-driven architecture, platform-independent model and platform-specific model | Client/server, publish/subscribe, master/slave, arbitrary control models within component hull | Being Developed | Yes | Yes | Yes | Two reference Implementations, one based on CORBA (ACE/TAO) and one based on ACE only | Yes | Yes | yes, runs with RTAI-Linux and QNX | Yes | Yes | Yes |
| ERSP | 3 layers | | No | | No | Yes | Yes | Comm | Yes | No | No | No | Yes |
| Skilligent | | | Yes | No | Yes | Yes | | Comm. | Yes | No | Yes | | |
| Webots | Multiprocess architecture | | Can simulate dynamical device failure: (physical destruction, noise increase, etc.) | Yes | Yes | Yes | TCP Socket interface, Open Dynamics Engine, Ogre 3D | Comm. | Yes | Yes | No | No | |
| Irobotaware | Layered architecture | Publish/subscribe, Messaging | | Yes | Yes | | | Comm. | Yes | No | Yes | Yes | Yes |
| Carmen | 3T hybrid architecture | | | 2D simulator | Yes | No | Socket based, using TCP protocol, IPC | Yes | No | No | Yes | Yes | Yes |
| RoboFrame | Message-oriented publish/subscribe and shared memory communication mechanisms | Message-based | No | Yes | Yes | Yes | Socket based | No | No | No | Yes | Yes | No |
| Pyro | Architecture independent | | No | Yes | Yes | Yes | Socket based using TCP protocol, XML, SOAP, OpenGL, HTTP | Yes | Yes | No | No | Yes | Yes |

## 3.6. Detectando el entorno: Visión Artificial.

En un principio, las aplicaciones robóticas, dados sus propósitos y objetivos a cumplir, requerían de una interacción sencilla con su entorno por lo que se empleaban sensores muy básicos para detectar ciertos estímulos de interés como sensores de proximidad o detectores de presencia. Sin embargo, el imparable desarrollo tecnológico ha desembocado en una mayor exigencia para los robots, lo que conlleva unas necesidades de interactuación superiores.

Actualmente es posible encontrar robots que atrapan objetos al vuelo, otros son capaces de correr por entornos hostiles (véase los desarrollos de Boston dynamics), incluso algunos tratan de hacer sombra a los futbolistas profesionales (robocup). De estos ejemplos, se deduce que la sensorización aplicada no es suficiente para la rapidez y cantidad de parámetros que deben analizar todos estos sistemas, siendo necesario dotar a los sistemas robóticos de capacidad para “ver” su entorno.

Según [González, Martínez y Co, 2006] [9] la visión artificial es una disciplina que engloba todos los procesos y elementos que proporcionan ojos a una máquina, pudiéndose decir que:

*La visión artificial o comprensión de imágenes, describe la deducción automática de la estructura y propiedades de un mundo tridimensional, posiblemente dinámico, bien a partir de una o varias imágenes bidimensionales de ese mundo. Las estructuras y propiedades del mundo tridimensional que se quieren deducir en visión artificial incluyen no sólo sus propiedades geométricas, sino también sus propiedades materiales.*

La visión, tanto para un ser humano como para un sistema artificial, consta principalmente de dos fases: captar una imagen e interpretarla. Centrándonos en un sistema robótico, la captación se realizará mediante un sistema de visión ya sea una cámara o un compendio de ellas, mientras que el procesamiento y tratamiento de la información se realizará por medio de software. Actualmente existen gran multitud de opciones para tratar las imágenes o videos capturados.

De entre las posibles soluciones disponibles actualmente destacan:

### 3.6.1. Matlab.

El programa matemático Matlab, entre la multitud de toolboxes integradas, presenta una destinada a visión artificial. Al igual que el resto de toolboxes, éstas se muestran en forma de funciones específicas y bloques de simulación para Simulink. El único inconveniente es la compatibilidad de plataformas. De este modo, si nuestra aplicación se ejecuta sobre una plataforma MAC o Windows, ésta será una opción a tener en cuenta dada la cantidad de manuales y comodidades integradas por este programa ampliamente utilizado en ingeniería.

9 [González, Martínez y Co, 2006] Técnicas y Algoritmos básicos de Visión Artificial. A.González, F.Javier Martínez, Alpha V Pernía, F.ALBA Y colabores. Universidad de La Rioja. 2006.

### 3.6.2. OpenCV.

Las siglas de OpenCv proceden de Open Computer Vision. De su nombre, se deduce que este software no supone sino un conjunto de librerías de visión por computador. Entre sus características figura la compatibilidad multiplataforma y cantidad de lenguajes de programación compatibles. Esta biblioteca de funciones contiene un conjunto de procedimientos, similares a los implementados en Matlab y otro software de visión, mediante los cuales se puede desde abrir una imagen capturada a obtener coordenadas, detectar objetos y realizar tareas de tracking o seguimiento. Señalar que ROS integra un paquete de compatibilidad con OpenCV dada su gran aceptación en el campo de la visión artificial.

### 3.6.3. PCL.

Acrónimo de Point Cloud Library. Se define como proyecto de código libre a gran escala para el procesado de imágenes en 2D/3D y los conocidos point clouds. Este framework contiene una amplia y variada cantidad de algoritmos entre los que figuran procesos específicos de filtrado, estimación de parámetros, reconstrucción de superficies, registro de eventos, segmentación… En cuanto a términos de licencia, dada su característica Open Source, las librerías PCL se encuentran bajos los términos de una licencia BSD de tres cláusulas, siendo gratuita para uso comercial e investigador. Al igual que OpenCV, uno de sus mayores competidores (o colaboradores, según la aplicación), es multiplataforma (Mac, Linux y Windows) y sus librerías quedan definidas en módulos compilables individualmente que permiten reducir el tamaño de las mismas, adaptándolas a las necesidades del desarrollo. Por último, queda definir la estructura típica de este software, un *point cloud*, o nube de puntos, es una estructura empleada para almacenar una colección de puntos multidimensionales y que permite representar datos tridimensionales. Así, por ejemplo, para representar un point-cloud tridimensional, los puntos son usualmente x, y, z lo que coincide con las coordenadas geométricas empleadas habitualmente para describir un espacio geométrico. Este tema será abordado con mayor profundidad en futuros capítulos.

# CAPÍTULO 4

# DEFINICIÓN DE LOS ELEMENTOS HARDWARE DEL PROYECTO.

## 4.1. Introducción.

La finalidad del presente capítulo abarca la descripción de todos los elementos hardware utilizados, con una profundidad suficiente que permita la emulación de los experimentos que constituyen la integridad y razón de ser de este Trabajo Fin de Grado, así como permitir la elaboración de futuros desarrollos.

En aras de la legibilidad y organización, este capítulo se divide en diversos apartados centrados en desglosar el abanico de características de que goza cada uno de los elementos constituyentes. Se comenzará así definiendo la piedra angular del proyecto, el robot manipulador antropomórfico Shadow Hand. A continuación, se proseguirá con la descripción de los distintos sistemas auxiliares utilizados, como el sistema de visión artificial constituido por una cámara Kinect. En cuanto a los ordenadores, puesto que se consideran material básico fundamental de disposición imperativa y obligada, no se incluirán en estas descripciones.

## 4.2. Robot Manipulador Shadow Hand.

En este apartado se realiza una descripción del robot manipulador Shadow Hand adquirido por la UPCT en 2011. Para su elaboración se han consultado los distintos manuales facilitados por el grupo de investigación que adquirió el robot, NEUROCOR. [10] [11] [12] [13]

[10] Shadow Cartagena Hand, Robert Warburton & Hugo Elias. 2011.
[11] Shadow C6M2 Hand – user manual. The shadow Robot Company Ltd. 2009
[12] Robot Control Software API. The Shadow Robot Company, Ltd. 2009
[13] Shadow Robot Software System. The Shadow Robot Company, Ltd. 2009

### 4.2.1. Modificaciones especiales del modelo Shadow C6M2-UPCT.

*Ilustración 13. Robot Shadow Hand*

La mano adquirida por el grupo de investigación del departamento de Sistemas y Automática es una versión modificada del modelo *Shadow C6M2*. La razón para la modificación de este robot manipulador de tres dedos fue la necesidad de controlar de forma externa y directa los motores mediante señales lógicas. Así, el firmware instalado cuenta con un modo para las tarjetas controladoras que permite controlar los distintos puentes en H por medio de señales externas.

Para lograr este comportamiento se integraron cuatro tarjetas controladoras, actuando una de ellas como maestro de los otros tres esclavos. Se tiene así que; mientras la tarjeta principal controla a las secundarias, cada una de estas últimas se encarga de gobernar tres tarjetas de control de motores. En la siguiente imagen, que corresponde a una vista de la base del antebrazo, se puede observar esta estrategia de control así como la distribución de las tarjetas:

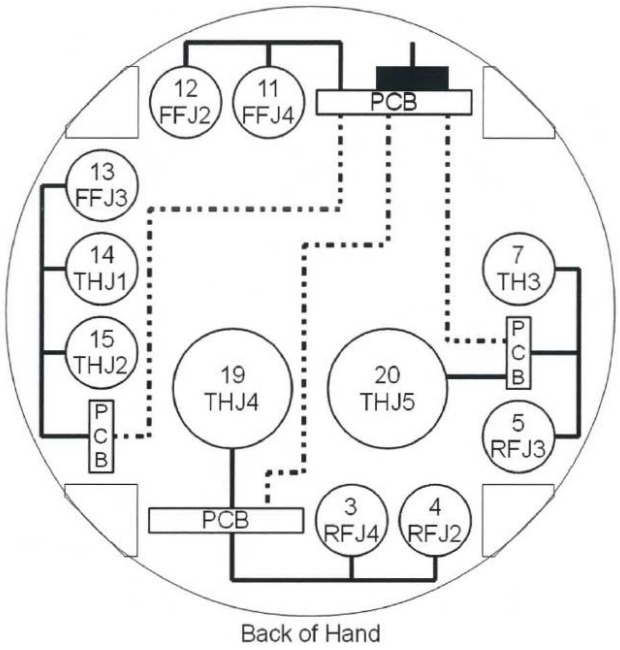



*Ilustración 14. Distribución de las tarjetas controladores en la Shadow Hand.*

Por diversos motivos, la mano no presenta el total de dedos disponibles para el robot (cinco y una muñeca articulada). En su lugar, se agenció un robot con tres dedos: pulgar, índice y anular. Aunque este hecho imposibilita realizar agarres de gran precisión, si presenta la configuración mínima requerida para lograr agarres estables menos ambiciosos que involucren geometrías más sencillas.

Como se puede apreciar en la siguiente figura, los motores que articulan la muñeca tampoco se encuentran presentes:

**Hardware layout**

There are 11 motor nodes in the system. The complete list is:

| Sorted by node | |
|---|---|
| Node name | Joint |
| smart_motor_4601 | MFJ4 |
| smart_motor_4602 | MFJ2 |
| smart_motor_4603 | RFJ4 |
| smart_motor_4604 | RFJ2 |
| smart_motor_4605 | RFJ3 |
| smart_motor_4606 | LFJ5 |
| smart_motor_4607 | THJ3 |
| smart_motor_4608 | LFJ3 |
| smart_motor_4609 | LFJ2 |
| smart_motor_4610 | LFJ4 |
| smart_motor_4611 | FFJ4 |
| smart_motor_4612 | FFJ2 |
| smart_motor_4613 | FFJ3 |
| smart_motor_4614 | THJ1 |
| smart_motor_4615 | THJ2 |
| smart_motor_4616 | MFJ3 |
| smart_motor_4617 | WRJ2 |
| smart_motor_4618 | WRJ1 |
| smart_motor_4619 | THJ4 |
| smart_motor_4620 | THJ5 |

| Sorted by joint | |
|---|---|
| Joint | Node name |
| FFJ2 | smart_motor_4612 |
| FFJ3 | smart_motor_4613 |
| FFJ4 | smart_motor_4611 |
| LFJ2 | smart_motor_4609 |
| LFJ3 | smart_motor_4608 |
| LFJ4 | smart_motor_4610 |
| LFJ5 | smart_motor_4606 |
| MFJ2 | smart_motor_4602 |
| MFJ3 | smart_motor_4616 |
| MFJ4 | smart_motor_4601 |
| RFJ2 | smart_motor_4604 |
| RFJ3 | smart_motor_4605 |
| RFJ4 | smart_motor_4603 |
| THJ1 | smart_motor_4614 |
| THJ2 | smart_motor_4615 |
| THJ3 | smart_motor_4607 |
| THJ4 | smart_motor_4619 |
| THJ5 | smart_motor_4620 |
| WRJ1 | smart_motor_4618 |
| WRJ2 | smart_motor_4617 |

*Ilustración 15. Motores presentes en Shadow-C6M2-UPCT.*

No obstante, se puede apreciar que el pulgar presenta más grados de libertad que el resto de dedos, concretamente una orientación más.

En el manual [10] se puede encontrar información adicional sobre la configuración y uso de señales lógicas externas, así como del procedimiento a seguir para configurar los modos de alta impedancia necesarios para este control. En caso de desear modificar o acceder al hardware de estas tarjetas, también es posible localizar los planos de las tarjetas a modificar, incluyéndose pautas para realizar reparaciones básicas de los dedos.

### 4.2.2. Descripción del robot.

A pesar de las modificaciones descritas en el apartado anterior, gran parte de las funcionalidades permanecen intactas y la forma de control habitual inalterada. Como se citó anteriormente, aunque las alternativas de control se ven ampliadas por las modificaciones solicitadas, únicamente se abordará la descripción y manejo del robot desde las perspectivas habituales, esto es, usando un ordenador con un software que administra las tareas de control y comunicación ya sea mediante API's o ROS. Este planteamiento queda justificado por la temática y objetivos del proyecto.

Todas las operaciones realizadas por la mano son enviadas desde el ordenador de control por medio de mensajes a través de un bus CAN. Concretamente, esta mano, al constar de tres dedos y carecer de las articulaciones de muñeca y palma, dispone de 13 articulaciones controlables:

- 4 articulaciones en cada dedo.
- 5 articulaciones en el pulgar.

La nomenclatura de los dedos se ha establecido acorde a los estándares médicos. Se tiene así que; las articulaciones más próximas a las yemas de los dedos reciben la denominación de "articulaciones distales". Mientras que, las más cercanas a la zona correspondiente al metacarpo se denominan "articulaciones proximales", estableciéndose el término de "articulaciones medias" para las restantes. Así, las tres primeras articulaciones se encargan de realizar el movimiento de flexión/extensión, mientras que la cuarta situada en la zona metacarpial se encarga de los movimientos de abducción y aducción. En el pulgar existe una quinta articulación que permite la rotación sobre su eje.

En cuanto a la métrica de los desplazamientos, se ha establecido como lógica positiva aquella relativa a movimientos encargados de extensión, mientras que se consideran negativos los referidos a desplazamientos de flexión.

La medición de los desplazamientos o rotaciones sufridas por cada articulación se consigue usando sensores de efecto Hall, los cuales generan una variación de voltaje dependiente del ángulo desplazado. Por este motivo, la salida del sensor es sinusoidal, hecho que obliga a definir el rango de acción o medición como [0, 180º]. Aunque esta restricción se integra cómodamente, sin mayores problemas en la mayoría de articulaciones, dado el amplio rango de movimiento de la articulación 5 del pulgar, han sido dispuestos dos sensores de efecto Hall para realizar la medición, distanciados 90º entre sí.

### 4.2.3. Perfil Mecánico.

#### *4.2.3.1. Dimensiones.*

El modelo del robot Shadow Hand C6M ha sido diseñado tratando de emular con gran precisión el perfil de una mano humana media. De este modo, el antebrazo, en el que se sitúan todos los motores, presenta una longitud similar a la de un antebrazo humano, siempre y cuando se desprecie su grosor. El conjunto de medidas puede observarse en la siguiente tabla facilitada por la empresa:

| | |
|---|---|
| Finger length<br>from tip of finger to middle of knuckle | 102mm |
| Thumb length | 102mm |
| Palm length<br>from middle knuckle to wrist axis | 99mm |
| Palm thickness | 22mm |
| Palm width | 84mm |
| Thumb base thickness | 34mm |
| Forearm<br>base to wrist axis | 208mm |

*Ilustración 16. Dimensiones del modelo C6M.*

Estas mismas dimensiones se pueden encontrar expresadas sobre plano, junto al rango de movimiento cinemático de cada articulación, en la siguiente figura:

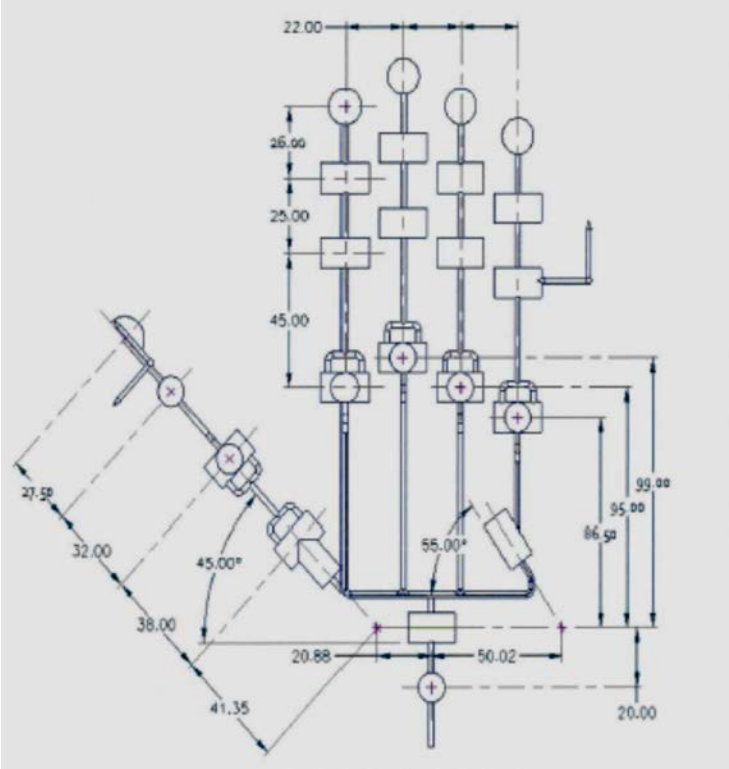



*Ilustración 17. Rango de movimientos y dimensiones del modelo C6M.*

### *4.2.3.2. Otras variables de interés: peso, velocidad y materiales de fabricación.*

La mano completa del citado modelo tiene un peso promedio próximo a los 4,2 Kg. Sin embargo, las modificaciones y reducción de articulaciones han reducido el peso del modelo dispuesto en el laboratorio a 3,6 Kg.

En cuanto a la velocidad de movimiento, ésta es relativa a cada articulación, aunque es posible hablar de una velocidad media global aproximada a la mitad del tiempo de reacción humana. Esto se traduce en 0,2 segundos para flexionar totalmente una falange.

Por último, los materiales empleados en el diseño han sido principalmente: acetileno, aluminio, policarbonato, metal y poliuretano.

### 4.2.4. Control y actuación.

Todo el hardware del robot se alimenta de una fuente de 49 Voltios de Continua. Puesto que no presenta sistemas de almacenamiento propios, con el hardware se encuentra una fuente de corriente que se encarga, además de proporcionar alimentación, de realizar la conversión de 230 Voltios alternos a valores adecuados para trabajar con el robot.

En cuanto al control, cada una de las articulaciones del robot se encuentra acoplada a un motor situado en el antebrazo. La conexión entre el motor y la articulación ligada se realiza por medio de dos tendones. Aparte del propio motor, cada unidad motora incluye los drivers electrónicos necesarios, un microcontrolador, sensores y células de carga. El siguiente esquema muestra el flujo de la información de control y las relaciones que se establecen para lograr que los comandos enviados desde el ordenador de mando se vean reflejados en el movimiento de las distintas articulaciones del robot.

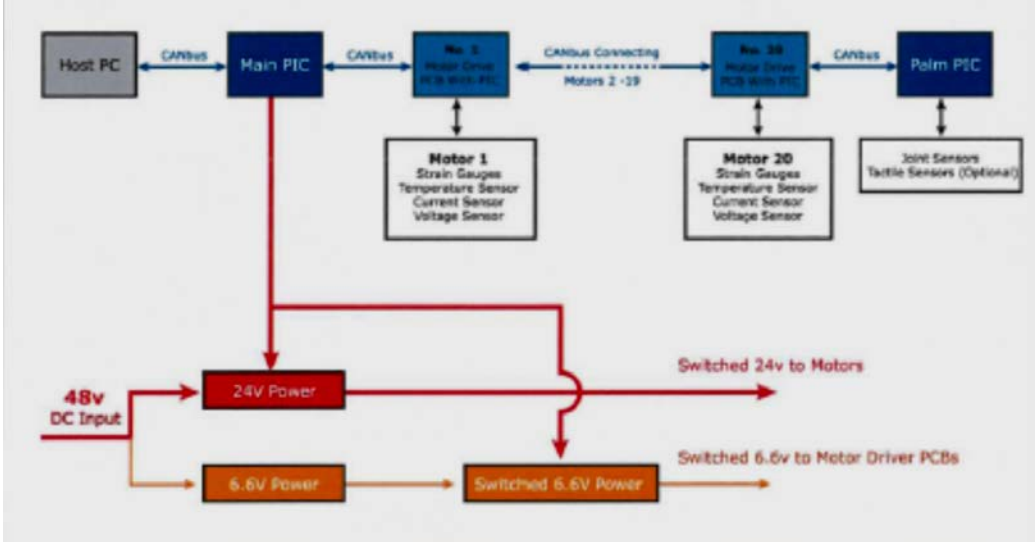



*Ilustración 18. Esquema de comunicaciones de la Shadow Hand.*

El bus CAN se conecta con el robot a través de la placa base, encargada de gestionar la sensorización del robot situada en la base del antebrazo. Esta placa se comunica con todos los convertidores analógico-digitales de los sensores de interés para el usuario (táctiles, posición de las articulaciones, temperatura…) por medio de SPI (Serial Peripheral Interface). Además, esta tarjeta integra sus propios convertidores ADC que se emplean para las mediciones de la muñeca, y articulaciones inferiores del pulgar (3, 4 y 5).

### 4.2.5. Sensorización y Calibración.

El modelo C6M presenta una gran cantidad de entradas y salidas. Por diversos motivos de diseño, la empresa decidió denominar, a pesar de la confusión que pudiese ocasionar, a todas éstas "interfaces", sensores. Antes de proseguir, recordar que la nomenclatura, dentro de cada dedo, se rige acorde a principios médicos, existiendo así falanges proximales, medias y distales. Así mismo, la arquitectura de construcción parte de la misma base para todos los dedos, salvo el pulgar, dada su amplitud de movimientos y número de articulaciones.

En la siguiente imagen puede observarse con mayor detalle el tipo de sensores y cableado establecido en el robot. Puesto que la arquitectura de todos los dedos, a excepción del pulgar, es análoga; se ha dispuesto un esquema de cada tipo de conexionado y distribución:

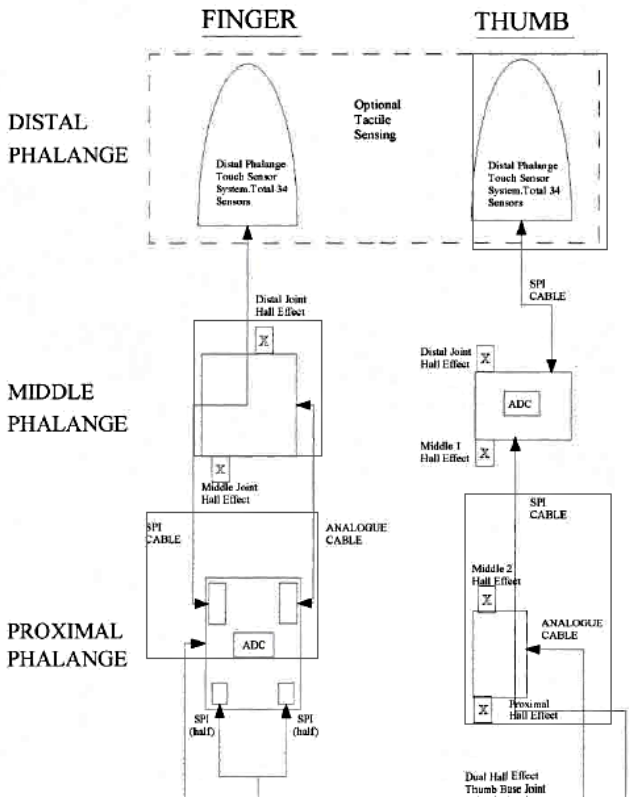



*Ilustración 19. Esquemas de sensores y cableado de los dedos.*

A pesar de seguir la pauta médica ya comentada, internamente, cada dedo presenta una nomenclatura propia para las articulaciones que integra. De este modo, en el sistema, el nombre de los sensores está constituido por 4 siglas. Las dos primeras hacen referencia al dedo del que forma parte el sensor, distinguiéndose entre seis opciones posibles:

- FF. Del inglés First Finger, se refiere al dedo índice.
- MF. Del inglés Middle Finger, alude al dedo corazón.
- RF. Del inglés Ring Finger, corresponde al dedo anular.
- LF. Del inglés Little Finger, da nombre al meñique.
- TH. Del inglés Thumb, contempla el pulgar.
- WR. Del inglés Wrist, hace alusión a la muñeca.

Por otro lado, las dos siguientes siglas hacen referencia a la articulación, siendo la primera de ellas una j, del término anglosajón *joint*, y la siguiente el número correspondiente a la articulación. Para la numeración de las articulaciones se han considerado varias premisas:

- La articulación distal recibe el número 1.
- La articulación 2 corresponde a aquella que induce el movimiento de la falange media.
- La articulación 3 es la responsable del movimiento de flexión de la falange proximal. Sin embargo, para el pulgar, se encarga de inducir un movimiento lateral en la falange media.
- La articulación 4 provoca el desplazamiento de aducción en la falange proximal. Para el pulgar, se corresponde con el movimiento que permite oponer dicho dedo a meñique.
- La articulación 5 sólo existe en el pulgar y el meñique. En el meñique se corresponde con el movimiento de oposición antes comentado, y en el pulgar permite la rotación sobre su eje.
- Para los dedos (excluyendo el pulgar) existe una articulación virtual denotada con el número 0. Esta articulación integra el movimiento a nivel software de las articulaciones 1 y 2, siendo imposible acceder a éstas de forma individual por medio de software.

Para más información sobre la disposición sensorial y calibración de los mismos consultar las páginas de la 12 a la 15 de la referencia [11].

### 4.2.6. Sistemas de Seguridad de la mano C6M.

El firmware del robot implementa distintas funcionalidades de seguridad y prevención de daños. Se ha considerado que, dado el comportamiento excepcional que puede presentar el robot, es recomendable que éstas sean comentadas. Antes de comenzar, señalar que una forma bastante útil de conocer el estado del sistema consiste en ejecutar la utilidad "motor_status" que puede encontrarse en el escritorio del ordenador que se entregó con el equipo.

Existen así, una serie de valores que provocarán una parada o interrupción inmediata del proceso que esté ejecutando el sistema, ya que son claros indicadores de problemas graves:

| Cutout | Operation | Parameter | Limit |
|---|---|---|---|
| Temperature | A temperature sensor on the Smart Motor PCB is in thermal contact with the Smart Motor housing. If the measured temperature is greater than the max_temperature value, then the motor will be turned off until the temperature has fallen by 2°C. | max_temp | 70°C |
| Safe Force | The resolved tendon force is compared to this value, and if the value is exceeded, then the motor is turned off. | motor_safeforce | 32767 |
| Current | While the motor current exceeds this parameter, a counter is incremented. The value of this counter is subtracted from the motor duty cycle to throttle the motor. | max_current | 300 |

*Ilustración 20. Rangos de seguridad del modelo C6M*

### 4.2.7. Anatomía de la cadena de transmisión de los motores.

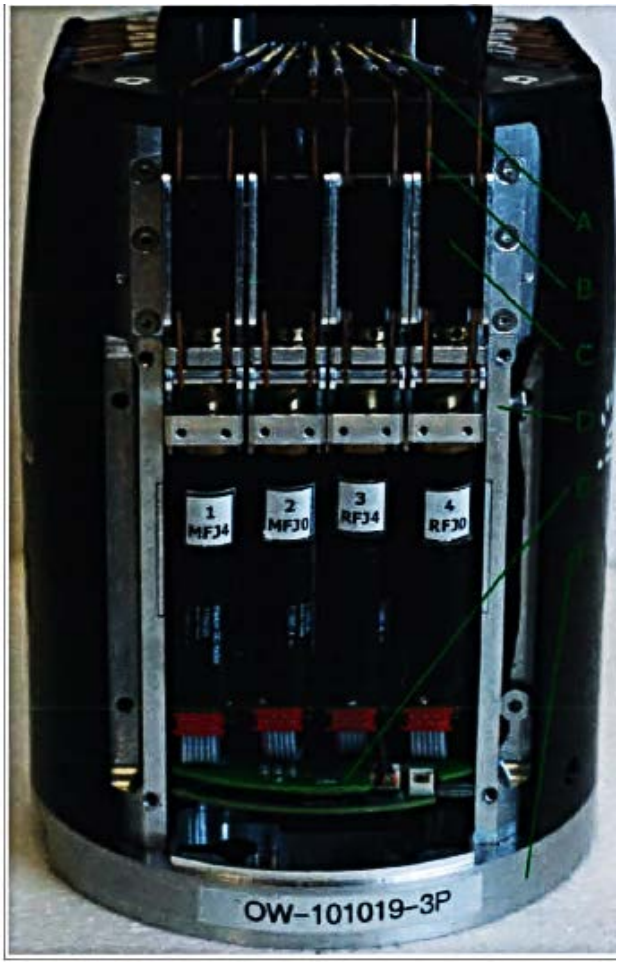



*Ilustración 21. Antebrazo del robot sin la carcasa.*

a) Recubrimiento del tendón. Proporciona protección a los tendones.
b) Tendón. Conecta a los motores con las articulaciones.
c) Tensor. Contiene un par de muelles que tratan de mantener y estabilizar la tensión en cada tendón.
d) Hueso. Forma parte de la estructura de la base.
e) PCB de la base. Se encarga de gestionar la alimentación y la comunicación CAN.
f) Base sólida para unir la mano a un brazo robótico.

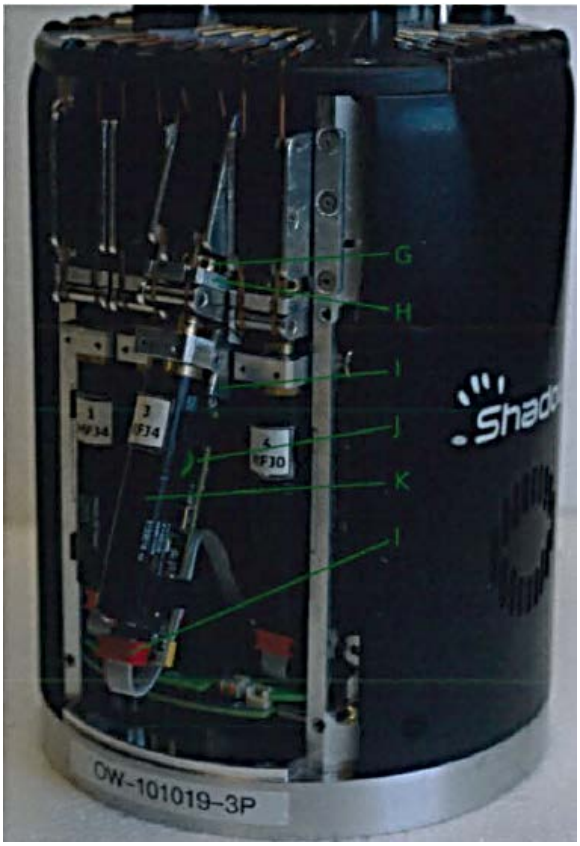



*Ilustración 22. Segunda ilustración del antebrazo sin carcasa.*

g) Bobina o carrete. A su alrededor se enrollan los tendones. Está constituida por dos partes unidas por un perno o pestillo. Anclándose un tendón en cada parte.
h) Envoltura del carrete. Contiene un motor, el carrete y las galgas extensiométricas.
i) Células de Carga. Existen dos de ellas en cada motor. Cada célula contiene una galga extensiométrica que mide la carga aplicada a cada tendón.
j) PCB, contiene el driver del motor, los sensores de corriente, temperatura y microcontroladores para la gestión de la alimentación.
k) Motor con escobillas de 3W con un ratio de transformación 131:1
l) Conector. Envía la información y alimentación desde la PCB base.

## 4.3. Sistema de Visión Artificial: cámaras industriales y de profundidad.

A la hora de escoger el sistema de visión artificial han sido analizadas, probadas y evaluadas distintas opciones. Los requisitos a cumplir, en un principio, por el sistema de visión artificial eran:

a) Detección de un objeto situado sobre una superficie plana.
b) Reconocimiento del color del objeto. Sirviendo éste como indicador de las dimensiones físicas del objeto.
c) Triangulación de las coordenadas del objeto en el entorno, respecto a la base del robot.

No obstante, a lo largo del desarrollo del proyecto y al analizarse distintas opciones, estos objetivos iniciales se vieron modificados y ampliados a:

a) Reconocimiento de posibles objetos de agarre dentro de un conjunto de varios objetos, conocidos y desconocidos para el sistema.
b) Triangulación del centroide de cada objeto que sea posible agarrar.
c) Determinación del centroide, en coordenadas tridimensionales, del objeto seleccionado para el *grasping*.
d) Adquisición de las dimensiones físicas necesarias para completar el agarre. Este punto presentará ciertas modificaciones que serán justificadas a lo largo del *Capítulo 6. Software Desarrollado.*

En un principio, se escogieron dos cámaras industriales del tipo uEye Industrial con objeto de generar un sistema estéreo que permitiese, entre otros aspectos, realizar la triangulación. No obstante, en cierto momento entraron en escena las cámaras de profundidad, hecho que obligó a replantear las posibilidades y conveniencia de invertir tiempo en cambiar de sistema. Puesto que este capítulo, sólo trata de abordar temáticas relativas a hardware, se presentarán ambos tipos de cámaras y concluirá realizándose una rápida comparación entre las posibilidades que cada una de ellas ofrece.

### 4.3.1. Cámara uEye Industrial USB2.0 SE.

Según señala el fabricante en su página web [14]:

*"La USB uEye SE es sin duda una cámara todoterreno. Pequeña, compacta y robusta, la primera cámara industrial de IDS lleva más de una década demostrando su valía. Gracias a la interfaz USB 2.0, la cámara es extremadamente flexible y ahorra espacio y costes puesto que la transmisión de datos y la alimentación se realizan a través de un solo cable. Las entradas y salidas digitales para el control del disparador y del flash están desacopladas ópticamente y procesan señales de hasta 30 V. Las cámaras pueden atornillarse por los cuatro costados y montarse en cualquier soporte. La junta especial del sensor garantiza la máxima protección contra el polvo. El filtro de la cámara puede cambiarse y limpiarse fácilmente."*

*Ilustración 23. Cámara uEye Industrial*

Destacando, entre sus características:

- Robusta: más de una década demostrando su eficacia en el sector industrial.
- Flexible: montaje universal (atornillable por los cuatro costados).
- Sólida: carcasa metálica compacta.
- Interfaz USB 2.0: fácil integración con funcionalidad plug&play.
- Disparador y flash desacoplados ópticamente: amplias funciones adicionales para múltiples aplicaciones en el ámbito industrial.
- Amplia gama de sensores: la cámara industrial USB 2.0 para todo tipo de aplicaciones.
- Filtro del sensor de imagen: protege contra la suciedad y sirve como filtro espectral de la luz incidente.
- Calibración y calidad 100% garantizadas: máximo rendimiento constante y alta fiabilidad.

14 HTTP://ES.IDS-IMAGING.COM/STORE/PRODUKTE/KAMERAS/USB-2-0-KAMERAS/UEYE-SE.HTML

A la hora de trabajar con esta cámara, el único inconveniente que se encuentra es la necesidad de instalar librerías de control y tratamiento de imágenes específicas. Esta salvedad, que bien podría ser, con un enfoque distinto, una ventaja, supone un lastre ya que obliga a desarrollar un script para captar las imágenes y tratarlas con el software facilitado por el fabricante, y acudir a un segundo script en el cual, mediante OpenCV u otro conjunto de librerías, se concluya el proceso que ha de cumplir el sistema de visión artificial.

### 4.3.2. Cámara de profundidad Kinect.

A diferencia de otras cámaras, las cámaras de profundidad se caracterizan por ser capaces de medir la profundidad de campo (DoF, depth of Field). Dichos dispositivos, además de generar matrices para informar de la coloración y puntos de la imagen, presentan una dimensión adicional que señala la profundidad a la que se encuentra cada punto discreto respecto del eje de coordenadas situado en el sensor. Uno de los primeros sensores conocidos de este tipo fue el revolucionario accesorio de juego de Microsoft, Kinect. No obstante, cada vez más se están desarrollando nuevos equipos de estas características.

El sensor Kinect, de la conocida plataforma de juegos Xbox, puede definirse como un dispositivo físico constituido por una cámara, un array de micrófonos y un acelerómetro así como una herramienta de software específica encargada de procesar color, profundidad y toda aquella información relacionada con el "*tracking de esqueletos*" (el objetivo principal del sensor es ser empleado en una consola).


*Ilustración 24. Cámara Kinect.*

Concretamente, en el interior del sensor se encuentra:

- Una cámara RGB que almacena información de los tres canales de datos con una resolución de 1280x960 píxeles.
- Un emisor de infrarrojo y un sensor infrarrojo para capturar la profundidad. Cuando el sensor captura los rayos reflejados en un objeto, convierte la medida en información de profundidad, en función de la distancia medida. De este modo, la cámara es capaz de generar imágenes de profundidad.
- Un array de 4 micrófonos.
- Un acelerómetro configurado para medir aceleraciones hasta 2G, donde G se corresponde con el valor de la aceleración de la gravedad.

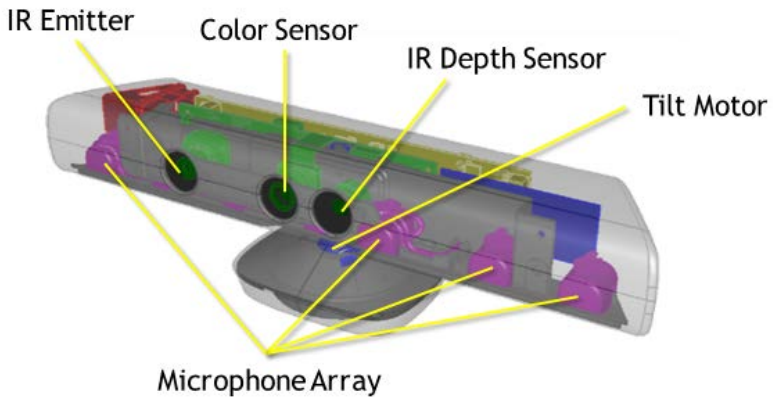



*Ilustración 25. Despiece del sensor Kinect.*

Aunque en el mercado existen cámaras equivalentes como la ASUS Z-CAM, se empleó ésta ya que era la única disponible para ser usada en el departamento de tecnología electrónica.

### 4.3.3. Comparativa.

Todos los objetivos señalados pueden ser satisfechos con ambos tipos de cámaras. Sin embargo, los costes de realización, en términos algorítmicos y de programación, son bastante diferentes. Por un lado, al usar sistemas tradicionales de visión estereoscópica se puede encontrar una gran variedad de guías y ayudas. Por otro lado, las cámaras de profundidad como Kinect, suponen un valor al alza cada vez más empleado en robótica, dada su velocidad de procesado y facilidad

para lograr triangulaciones. En cuanto a calidad de imagen, las cámaras de profundidad no son capaces de hacer frente al despliegue de potencial logrado por la electrónica de IDS.

A estos puntos debe añadirse la dificultad añadida que presenta el par estereoscópico uEye que posee el laboratorio; aunque los chips son idénticos (punto clave para visión estéreo), las resoluciones y calidad de las imágenes adquiridas no lo son. En casos como este, es necesario realizar un acondicionamiento por software para lograr dos imágenes de propiedades y características idénticas. Por este motivo, vistas las comodidades ofrecidas por las cámaras de profundidad, la limitación temporal y contando con la motivación de aprender a usar un software de visión artificial novedoso como PCL, se decidió traducir el desarrollo realizado con OpenCV a PCL.

# CAPÍTULO 5
# CONFIGURACIÓN DEL SOFTWARE Y PRIMEROS PASOS.

## 5.1. Introducción.

En este capítulo se abordarán todos los aspectos relativos a la configuración del software, al mismo tiempo tratará de proponer una serie de pautas para familiarizarse con el mismo, concluyendo con un conjunto de comandos que pueden resultar de gran utilidad y un solucionario de los principales errores que pueden acontecer. Puesto que a lo largo del desarrollo del proyecto se han empleado diversos recursos software, todos estos aspectos serán aplicados a cada tipo de software.

Este punto, aunque irrelevante para otro tipo de proyectos, ha sido uno de los principales lastres para el desarrollo del presente trabajo, siendo numerosos los problemas y altercados acaecidos durante su realización. Por ello, dada la amplitud y número de puntos de que debería constar, se ha decidido, por evidentes motivos de claridad y organización, articular el presente apartado como si de un informe de procesos y fases se tratase, quedando el desarrollo completo pertinente expuesto en anexos que puedan ser consultados en caso de interés para el lector.

Antes de comenzar, señalar que; dada la incompatibilidad del hardware del modelo que posee la Universidad Politécnica de Cartagena de la Shadow Dexterous Hand con ROS, se han debido realizar sockets para comunicar dos ordenadores que ejecutan distintas distribuciones de Linux. De este modo, a lo largo del presente capítulo, se abordarán dos distribuciones de Linux, una versión antigua de Debian sobre la que se encuentran instaladas las API's de control del robot, y un segundo ordenador ejecutando Ubuntu, con ROS, que controla el sistema de visión artificial.

Aunque la correcta instalación y configuración de Debian suele ser un proceso arduo y laborioso, en este caso, no se abordará en gran detalle ya que la empresa proporcionó el software ya instalado, siendo necesario ponerse en contacto con ellos si surgen necesidades de reinstalación o corrección de bugs.

En cuanto al segundo ordenador, se ha escogido Ubuntu como sistema operativo ya que es la plataforma en la que más avanzado se encuentra el desarrollo de ROS, consta de versión estable, siendo el resto de opciones aún

versiones Beta, o incluso Alfas, que pretenden estar adaptadas en un futuro, según se especifica en la página web oficial de ROS. [15]

Una prueba de ello se puede encontrar en la siguiente imagen, la cual muestra una tabla con todas las plataformas en las que se puede instalar la versión Hydro de ROS, desarrollada a principios de 2014. Como se puede comprobar, la versión aprobada se encuentra en Ubuntu, mientras que el resto de opciones son puramente experimentales llegando algunas a encontrarse en fase Alfa de desarrollo.

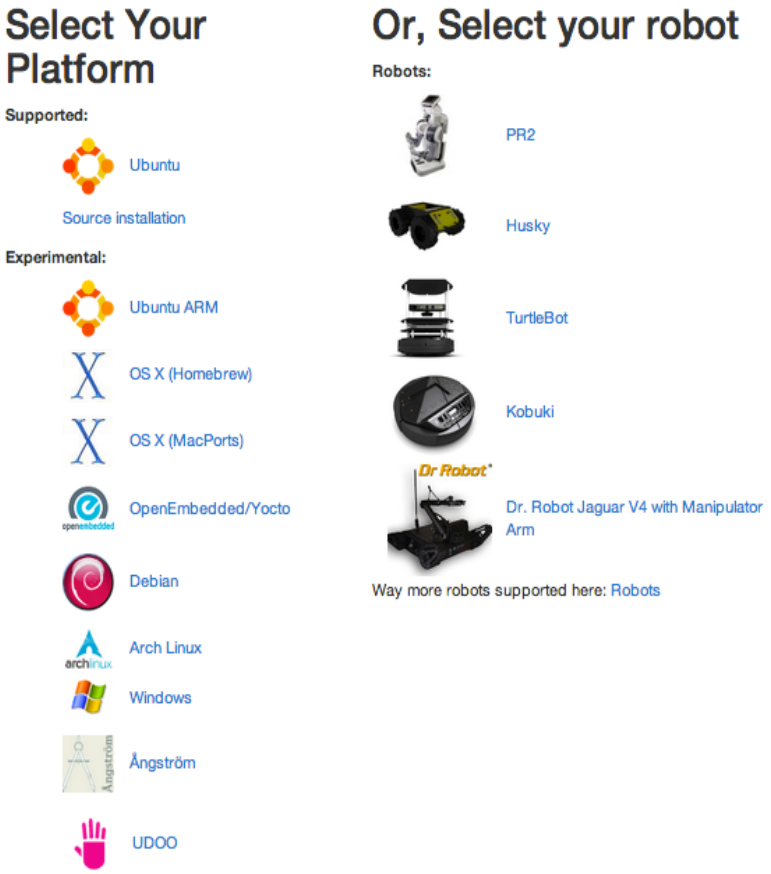



*Ilustración 26. Tabla de posibilidades de instalación de ROS.*

Así, sin más preámbulos, se da por concluida esta introducción e inicia el desglose de los distintos aspectos, relativos a la configuración, que han debido abordarse; desde configuraciones básicas en Debian hasta reinstalaciones completas de ROS y Ubuntu.

15 HTTP://WIKI.ROS.ORG/

## 5.2. Ubuntu.

### 5.2.1. Introducción.

Por todos los motivos citados en la introducción, se ha seleccionado Ubuntu como distribución de Linux para el trabajo con ROS y por ende con el sistema de visión artificial. Aunque resultaría conveniente que trabajase también con el robot, éste es totalmente incompatible, por lo que es necesario desarrollar soluciones adicionales que serán expuestas más adelante.

Tal y como se expuso en el capítulo 3 (*3.3* Linux como núcleo del sistema), Ubuntu es un sistema operativo basado en Linux y que se distribuye como software libre, el cual incluye su propio entorno de escritorio denominado Unity. El nombre proviene del concepto africano ubuntu, que significa "humanidad hacia otros" o "yo soy porque nosotros somos". También es el nombre de un movimiento humanista sudafricano. Ubuntu aspira a impregnar de esa mentalidad al mundo de la informática. El eslogan de Ubuntu, “Linux para seres humanos”, resume una de sus metas principales: hacer de Linux un sistema operativo más accesible y fácil de usar.

Así, este sistema operativo se encuentra orientado al usuario novel y promedio, pudiendo ser uno de los sistemas basados en Linux que más se centra en la facilidad de uso y en mejorar la experiencia de usuario.

Por último, antes de abarcar los distintos procesos de instalación y configuración del sistema, es imperativo hacer referencia a la filosofía de Ubuntu. Como proyecto Open-Source, éste se articula y construye sobre unos pilares básicos que desembocan en una filosofía propia y característica del sistema:

- La libertad de uso del programa, con cualquier propósito.
- La libertad de estudiar cómo funciona el programa y modificarlo, adaptándolo a cada necesidad.
- La libertad de distribuir copias del programa, con lo cual se puede ayudar al prójimo.

- La libertad de mejorar el programa y compartir esas mejoras con los demás, de modo que toda la comunidad se beneficie.

### 5.2.2. Instalación y Configuración del sistema.

Actualmente dada la cantidad de usuarios, la actividad de la comunidad y la cuantía de programas de emulación de software existentes, es posible usar Ubuntu en nuestro ordenador empleando diversos medios. De entre todos ellos, se puede señalar que las principales opciones son:

- Instalación Directa y Nativa.
- Instalación mediante Xubi.
- Emulación en máquina Virtual.

Se recomienda encarecidamente probar el sistema, preferiblemente en máquina virtual si se es usuario novel, antes de tratar de realizar una instalación completa con objeto de verificar las incompatibilidades que puedan acontecer. Si bien es cierto que las versiones de Ubuntu son innumerables, muchas de ellas presentan problemas con ciertos tipos de hardware. Prueba de ello son la gran cantidad de foros abiertos y listas de compatibilidad creadas por esta trabajadora comunidad.

En cuanto a la instalación en máquina virtual, dada la sencillez del proceso, se citarán únicamente algunas opciones del software que puede emplearse para dicha emulación:

- Vm Ware.
- Virtual BOX (gratuito).
- Parallels (Muy recomendado, sólo MAC).

Por otro lado, se ha mencionado Xubi, ésta es una estrategia de configuración que se encuentra a caballo entre la instalación nativa y la emulada. Esta aplicación, que se puede encontrar en la propia imagen de instalación, permite instalar Ubuntu como si de una aplicación de Windows se tratase. Aunque este TFG se inició con este sistema, totalmente recomendable para realizar el

proceso de familiarización, pronto tuvo que cambiarse, dada las limitaciones de rendimiento existentes, y adquirir un sistema hibrido.

En el *Anexo 1. Instalación de Ubuntu*, es posible encontrar más información del proceso de instalación nativa. Centrado éste principalmente en cómo realizar la instalación híbrida en un sistema con Mac OS X.

### 5.2.3. Primeros Pasos.

Una vez instalado el sistema en el equipo de trabajo y antes de proseguir con el uso de ROS conviene relacionarse con el mismo, ya que aunque a primera vista pueda parecer sencillo e intuitivo, presenta una amplia y diversa variedad de diferencias que ocasionarán numerosos quebraderos de cabeza a los usuarios noveles, y no tan noveles.

A pesar de que se pueden encontrar numerosos tutoriales y manuales de uso, en el *Anexo 2. Primeros Pasos en Ubuntu* se ha incluido una breve guía de manejo orientada al desarrollo del presente proyecto. Así mismo, al final de dicho anexo puede encontrarse una relación de comandos seleccionados que pueden resultar de gran interés para todo usuario de Linux.

Aunque el procedimiento de instalación de Ubuntu puede no parecer muy complejo, la dificultad de éste se ve acrecentada exponencialmente con cada pequeño error que surja relacionado con el hardware o modificación inesperada del software. De ahí, que se realice tanto hincapié en conocer el sistema y no realizar acciones "por probar" ya que esta medida, que en otro tipo de sistemas puede resultar satisfactoria, en Linux si algo puede fallar, fallará.

## 5.3. ROS: Instalación y configuración de PCL y Shadow Hand.

### 5.3.1. Introducción.

De entre el amplio elenco de middlewares presentados a lo largo del *CAPÍTULO 3. SOFTWARE APLICADO A LA ROBÓTICA*, y no presentados, se ha escogido ROS para trabajar. Del compendio de motivos que pueden enumerarse para justificar esta selección, destaca la necesidad de comunicar varios sistemas individuales para que actúen como si de un solo ente se tratase, y por otro lado, la hirviente curiosidad por probar y formar parte de la comunidad de desarrolladores que se sirve de este revolucionario sistema como medio de trabajo diario.

Otro de los principales motivos de esta elección se puede localizar en el fuerte apoyo que ha dado, y está dando, la comunidad de desarrolladores a este middleware. Este hecho, en gran parte, queda justificado por todas las herramientas que incorpora e integración del mismo en los entornos Linux, especialmente Ubuntu. Un claro aval del software se localiza en el gran conjunto de robots que ya han sido integrados con total funcionalidad en ROS, como puede observarse en la relación de robots mostrada en el anexo 3.

Una segunda prueba, de la aceptación de ROS, procede de la mano de los investigadores y desarrolladores. Según Willow Garage, a pesar de no contar con estadísticas sobre el número real de usuarios, debido al carácter Open-Source del middleware, si se puede garantizar el creciente número de visitas a la página, la cual alcanzó un máximo en 2013, con el lanzamiento de groovy, de 55.000 mensuales. Así mismo, el número de repositorios publicados e indexados ha crecido exponencialmente desde su aparición en el paradigma de la robótica:

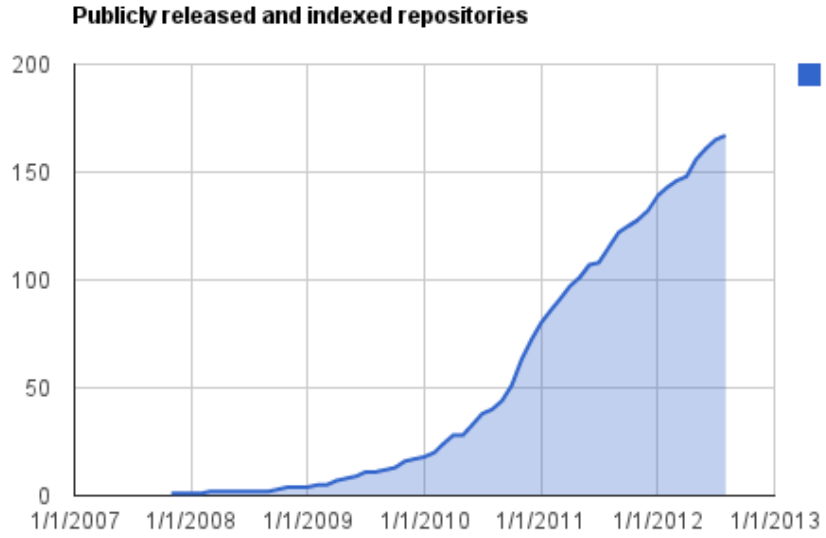



*Ilustración 27.Crecimiento de ROS.*

El siguiente motivo a citar reside en sus características. Tal y como señalaba [García, 2013] [16] ROS proporciona una serie de funcionalidades de gran interés:

- Abstracción del hardware.
- Acceso a los componentes del hardware.
- Mecanismo de paso de mensajes entre procesos.
- Mecanismos de construcción, pruebas y logging.
- Gestión de los componentes del sistema.
- Simuladores/ visualizadores…
- Librerías matemáticas.
- Librerías de Geometría.
- Librerías de reconocimiento de imágenes
- …

[16] HTTP://ES.SLIDESHARE.NET/VICEGD/DESARROLLO-ROBTICO-ROBOT-OPERATING-SYSTEM-ROS

Además, los principios de diseño del software también resultan muy interesantes, manifestándose rápidamente conforme se avanza en el manejo del sistema:

- Open- Source.
- Multiplataforma y multilenguaje.
- Procesamiento Distribuido con diseño modular.
- Apoyo y soporte de componentes externos:
  - OpenCV
  - Eigen
  - ODE+Gazebo
  - KDL (librerías de cinemática y dinámica).
  - TREX (planificación de alto nivel).
  - PCL ( Soporte para Point Cloud Library)
  - …

Una vez justificada su selección, puede ser interesante revelar más información sobre este middleware. ROS fue desarrollado originariamente por el Stanford *Artificial Intelligence Laboratory,* en 2007, bajo el nombre de *Switchyard,* con objeto de proporcionar soporte al proyecto *Stanford AI Robot STAIR.* Sin embargo, a partir de 2008, el proyecto continuó siendo desarrollado por Willow Garage, un instituto/incubadora de investigación robótica con más de 20 instituciones colaboradoras. El principal objetivo de Willow Garage era propulsar los desarrollos robóticos no relacionados con proyectos militares.

Tras desarrollar siete versiones de este software:

- 22 July 2014 - Indigo Igloo
- 4 September 2013 – Hydro Medusa
- 31 December 2012 – Groovy Galapagos
- 23 April 2012 – Fuerte
- 30 Aug 2011 – Electric Emys
- 2 March 2011 – Diamondback
- 3 August 2010 – C Turtle
- 1 March 2010 – Box Turtle
- 22 January 2010 – ROS 1.0

En febrero de 2013, la administración de ROS pasó a ser jurisprudencia de Open Source Robotics Foundation. Siendo en agosto de 2013, según reveló en su blog, Willow Garage absorbida por Suitable Technologies, ambas propiedad del mismo fundador. [17] [18]

Como se puede comprobar, es posible encontrar y aglutinar gran cantidad de explicaciones y razonamientos que abarcan desde la comodidad ofertada por el middleware a la hora de embeber sistemas, hasta la activa comunidad de usuarios, que constituye el mayor servicio de soporte y ayuda posible. Sin embargo, este punto concluirá sus elogios hacia el middleware en estas líneas.

Una vez presentado el protagonista, se expondrá el modo en que ha sido desarrollado el presente punto. Al igual que ocurría con el punto anterior, con objeto de no colmar la paciencia del lector y agilizar la lectura de este TFG, este apartado ha sido subdivido en diversos apartados que tratan de cubrir los distintos aspectos que han resultado relevantes durante la realización del proyecto, limitándose la exposición del capítulo a un resumen del desglose detallado que puede encontrarse en los anexos correspondientes. Así, sin más preámbulos, la parte de ROS puede quedar sub-dividida en tres apartados básicos:

- Instalación y configuración.
- Primeros Pasos.
- Manejo del simulador de la Shadow Hand en Gazebo.

### 5.3.2. Instalación y configuración.

Aunque la instalación de ROS podría resumirse en abordar la dirección del tutorial presente en la página oficial de ROS y copiar los comandos especificados por este equipo de desarrolladores, para este caso particular se deben realizar una serie de tareas adicionales. Entre este conjunto de configuraciones adicionales figura la instalación de los paquetes y herramientas necesarias para controlar el robot Shadow Hand de la empresa Shadow Company (controladores,

17 http://www.willowgarage.com
18 http://en.wikipedia.org/wiki/Robot_Operating_System

simuladores…), los paquetes de PCL que permiten emplear un sistema de visión artificial, controladores de la cámara a usar…

Puesto que el robot de que dispone el grupo NEUROCOR de la UPCT no es compatible con ROS, los paquetes de la mano Shadow Hand han sido instalados no sólo para conocer el modo en que se controlaría una mano equivalente compatible, sino también para probar cierto software en el simulador de la misma antes de aplicarlo al robot real. Como se verá más adelante, todo el código desarrollado, ha sido escrito de forma tal que su integración en ROS se reduzca a modificar dos funciones de 5 líneas de extensión. Esta estrategia de programación desarrollada se abordará con mayor amplitud durante el *Capítulo 6. Software Desarrollado.*

Así, tras finalizar con este proceso, se obtiene un simulador de la Shadow Hand empleando Gazebo, con un funcionamiento de calidad más que aceptable si se compara con el modelo real, y un conjunto de librerías que facilita embeber el código de visión artificial desarrollado en ROS. Para más información sobre el proceso consultar *el Anexo 4. Instalación y configuración de ROS (PCL y Shadow Hand).*

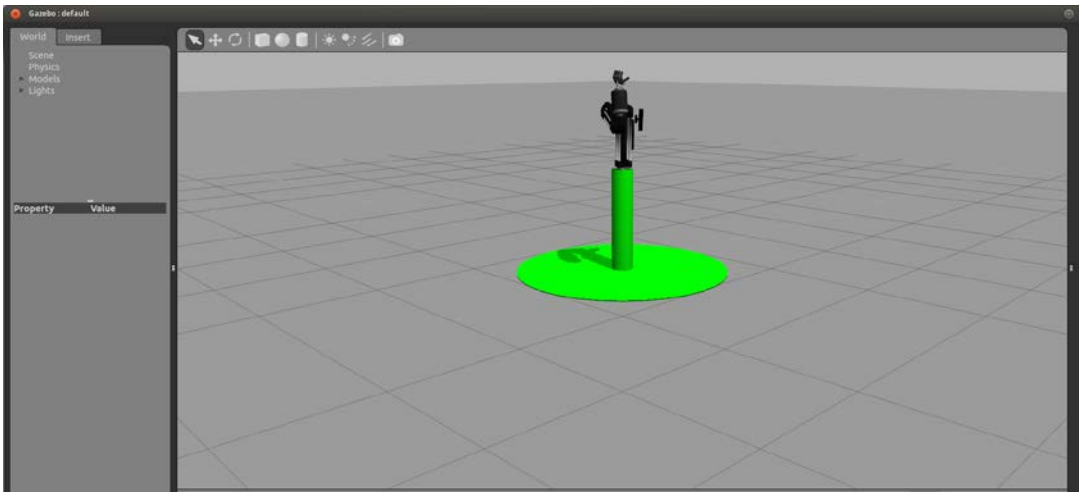



*Ilustración 14. Simulador Shadow Hand Gazebo en ROS.*

### 5.3.3. Primeros Pasos en ROS.

Dejando aparte todas sus características, si por algo destaca ROS es su escarpada curva de aprendizaje. Con ello no se pretende desanimar al emocionado usuario novel, pero si advertirle y prevenirle del arduo camino en el que se está introduciendo. Ningún usuario, exceptuando a los prodigios, puede pretender ser un profesional y dominar ROS en unas semanas, ni tan siquiera unos meses. Son muchos los conceptos a asimilar, abundante la filosofía que comprender y complejas las relaciones a desarrollar.

Por ello, en la página oficial se proporciona un amplio tutorial que trata de sobrevolar todo el sistema. Así mismo son incontables los foros existentes y preguntas realizadas en ellos. Siendo de estos incentivos e iniciativas de los que surge la idea de realizar una pequeña aportación a la comunidad por medio de la inclusión de un anexo que resuma los tutoriales consultados a lo largo del TFG y centrado en expresar de forma más cercana todos estos conceptos.

Realizadas estas consideraciones y sin ahondar en más detalles, se deja a modo de consulta opcional el resto de información relativa a este apartado en el *Anexo 5. Primeros Pasos en ROS*.

### 5.3.4. Manejo del simulador de la Shadow Hand en Gazebo.

En el *Anexo 4. Instalación y configuración de ROS (PCL y Shadow Hand),* entre otros paquetes, se han instalado un conjunto de archivos que permiten controlar el manipulador robótico antropomórfico Shadow Hand desde ROS. Además, este software incorpora un simulador, basado en Gazebo, que imita el comportamiento del robot real, incluyendo las publicaciones y topics del mismo, con la salvedad de introducirse una pequeña variación en el nombre de los topics para aquellas situaciones en las que se requiera emplear simulador y robot conjuntamente.

A la hora de desarrollar programas esta herramienta de simulación resulta de gran utilidad- Recordar que Gazebo se puede programar mediante ficheros .launch, los cuales fueron presentados en el apartado *Ejecución de programas y*

*visualización de datos: roslaunch y rqt_graph* del *Anexo 5. Primeros Pasos en ROS*. De este modo, mediante la programación de gazebo, es posible añadir más objetos a la escena de simulación con los que el robot pueda interactuar. Así, se permite realizar un debug de un programa de agarre, de grasping estable, de comunicación e interactuación con los patrones de una cámara Kinect…

A lo largo del *Capítulo 6. Software Desarrollado* se incluye un ejemplo, creado para el presente TFG, en el que se muestra cómo realizar la programación para generar un nuevo marco de escena en el que el robot Shadow Hand simulado interactúa con una cámara de profundidad. Podrá comprobarse como la configuración de Gazebo mediante archivos .launch, aunque proporciona una capacidad de virtualización muy elevada, es bastante compleja, especialmente si no se está relacionado con los lenguajes de sintaxis marcada, en especial YAML.

Se ha considerado el desarrollo de este punto de especial interés ya que las fuentes proporcionadas por Shadow pueden ser consideradas escuetas y carentes de estructura, aunque suficientes para desarrolladores experimentados. Por ello, en el *Anexo 6. Introducción al uso del simulador de Shadow* se proporcionan una serie de directivas que pueden servir de introducción a todos aquellos no familiarizados con el concepto.

## 5.4. Visión Artificial con PCL.

### 5.4.1. Introducción.

Antes de comenzar, señalar que este proyecto se inició tratando de integrar un sistema de visión artificial, constituido por un par de cámaras estereoscópicas, en OpenCV. Por lo tanto, la primera fase de esta parte del desarrollo anduvo involucrada en diversos aspectos del aprendizaje y manejo de OpenCV.

Sin embargo, la conferencia ROS-RM, realizada en la Universidad de Alicante, otorgó nuevos aires de innovación y renovó las motivaciones del trabajo. Tras la exposición realizada por Federico Tombari en este WorkShop; las técnicas y procedimientos que PCL proporciona a este paradigma particular de la robótica pasaron a ser consideradas y analizadas con sumo cuidado. Finalmente, una vez

concluido un compendio de pruebas de campo y revisados varios artículos, se decidió imprimir un giro en la parte de visión artificial.

Las pruebas realizadas se basaron principalmente en uso del hardware y procesamiento básico de imágenes. Los resultados obtenidos en ellas, pueden ser resumidos en:

- OpenCV requería uso de cámaras estéreo. De acuerdo a lo citado en el *CAPÍTULO 4. DEFINICIÓN DE LOS ELEMENTOS DEL PROYECTO*, estas cámaras, del tipo uEye Industrial, presentan un software propio de control que realiza parte del tratamiento gráfico, debiendo ser el restante implementado en OpenCV. Como elemento adicional, resultaba imperativo realizar un ajuste de imágenes ya que ambos modelos, aunque iguales, presentaban un desfase temporal de fecha de lanzamiento lo que es traducido en parámetros distintos en cuanto a resolución y calidad de imagen. En cuanto a su integración con ROS, es necesario realizar un wrapping del código desarrollado.

- PCL, por otro lado, al trabajar con nubes de puntos requiere de una cámara de profundidad, en este caso, la cámara Kinect de la conocida firma de terminales de juego Xbox. Como se verá más adelante, este tipo de cámaras no entrega una imagen como tal, sino que, por el contrario; las cámaras de profundidad entregan un archivo de *extensión pcd*. Los archivos de nubes de puntos se pueden definir como matrices de puntos tridimensionales, siendo la principal baza de estos sistemas conocer la distancia a la que se encuentra cada punto de la cámara.

De ambas comparaciones se extrae que; por un lado, OpenCV, aunque ampliamente utilizado, presenta ciertos problemas de aplicación focalizados principalmente en la configuración hardware. Por otro lado, PCL, *"menos conocido"* pero ascendiendo imparablemente hacia el liderazgo de los sistemas de visión, reducía toda la problemática a desarrollos software dependientes principalmente de la pericia del programador, siendo el principal problema

hardware que puede encontrarse, con esta tecnología, la resolución o limitaciones físicas del propio sistema de adquisición.

Dando por justificado este cambio en las directivas del trabajo, se pasará a introducir PCL con mayor profundidad. Para concluir, se expondrá su proceso de instalación, apartado *5.4.2 Instalación de PCL y drivers necesarios*, y añadirán unas referencias para comenzar a trabajar con él, apartado *5.4.3 Primeros pasos con PCL.*

Según F.Tombari, PCL, acrónimo de Point Cloud Library, puede entenderse como un proyecto hermano de OpenCV. Actualmente su mantenimiento y soporte se encuentra a cargo de una fundación filial sin ánimo de lucro conocida como Open Percepcion. No obstante, su concepción corrió a cargo del conocido grupo de desarrolladores *Willow Garage*. De entre sus características, destaca su licencia BSD de código libre.

Como ya se citó, estas librerías han desplegado un abanico de posibilidades de tal calibre que están desbancando al resto de opciones de visión artificial en diversos campos tecnológicos relacionados con la robótica: robots personales, coches autónomos (google), wearables…

Tal y como enunció este conferenciante, uno de los principales motivos que ha favorecido el despegue de PCL se localiza en las limitaciones que presentan los sistemas en 2D para gran cantidad de las aplicaciones desarrolladas actualmente, encontrando así un punto de apoyo en el grado de dificultad requerido por otros paquetes competidores que ofertan características 3D. Este hecho se ve respaldado y avalado por el grupo financiero que sustenta PCL, a diferencia de otras iniciativas de código libre, detrás de esta tecnología se encuentran multinacionales con sobrenombre dentro del panorama tecnológico de la época:


*Ilustración 15. Relación de empresas que mantienen económicamente a PCL.*

Así, todas estas características y apoyos de los que hace alarde PCL confluyen en un crecimiento exponencial que sitúa la cifra de contribuidores y usuarios de la tecnología en un valor superior a los 700.000 desarrolladores.

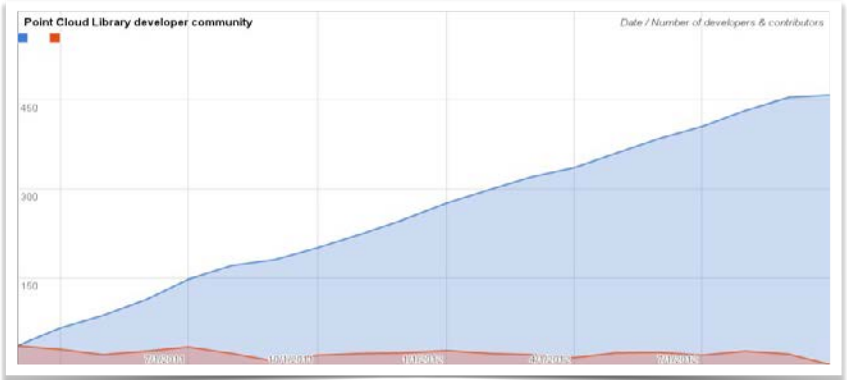



*Ilustración 16. Crecimiento PCL 2010-2013*

Por último, señalar que PCL, al haber sido diseñado por el equipo de Willow Garage, está ideado con la intención de ser integrado cómodamente en ROS, simplemente es necesario incluir la librería ros/ros.h y declarar los publicadores y subscriptores que interesen para la aplicación a desarrollar.

### 5.4.2. Instalación de PCL y drivers necesarios.

La instalación de PCL se resume, en la mayoría de casos, en instalar la versión necesaria mediante línea de comandos y los drivers necesarios para la cámara que se utilice, en caso de que Ubuntu no proporcione drivers genéricos suficientes. Para el caso en el que se haya instalado ROS, probablemente la versión que venga instalada de PCL, última estable compatible en el momento de lanzamiento de la distribución, será suficiente siendo posible saltarse los pasos de instalación y configuración de las librerías.

Aunque el procedimiento de instalación de PCL está totalmente detallado en la página oficial de la firma [Página Oficial de PCL] [19], para su uso con Kinect es necesario realizar una serie de configuraciones adicionales. Por ello, y al ser una parte del proceso de puesta marcha del presente proyecto, se han incluido todos los pasos necesarios a realizar en el *Anexo 7. Instalación de PCL y drivers necesarios.*

### 5.4.3. Primeros pasos con PCL.

El proceso de aprendizaje de PCL se encuentra descrito con total lujo de detalles y ejemplos de código en la sección de documentación de la página oficial, [PCL Tutorials] [20]. Por ello, en lugar de realizar una traducción del citado sitio web, el *Anexo 8. Primeros Pasos con PCL* ha sido reorientado para definir, con mayor detenimiento, ciertos conceptos y aspectos de PCL que pueden resultar de utilidad y ayuda durante el proceso de aprendizaje. Gran parte de estos conceptos, por no señalar su plenitud, fueron adquiridos durante la charla de Federico Tombari en la ROS-RM [F.Tombari. Sesion IV ROS-RM] [21] y asimilados a lo largo de las semanas de trabajo con esta tecnología.

19 [Página Oficial de PCL] http://pointclouds.org

20 [ PCL TuTORIALS ]http://pointclouds.org/documentation/tutorials/

21 [F.Tomabari. Sesion IV ROS-RM] Session IV: 3D Object Recognition in Clutter with the Point Cloud Library. Dr. Federico Tombari (University of Bologna-Italy, Italy / Open Perception, Inc., USA).

## 5.5. Configuración de Debian y puesta en marcha de la Shadow Hand.

En este apartado se cubrirá el proceso de configuración en Debian, especialmente la configuración de red ya que ocasionó numerosos problemas. Además, se desglosará el proceso de encendido y apagado del robot así como se enumerarán una serie de utilidades, proporcionadas por Shadow, que han sido usadas recursivamente para probar ciertos programas y configuraciones.

### 5.5.1. Configuración de redes en Debian.

La versión de Debian instalada en el equipo, proporcionado por Shadow, presenta ciertos problemas de integración. Por ello, se recomienda realizar todos los procesos de configuración por línea de comandos ya que la GUI (*Graphical User Interface*) no está totalmente comunicada con el sistema nativo, por motivos que no han podido solventarse.

De este modo, el único quehacer, a excepción de las configuraciones de sensores y PIDs de control que sean necesarias (el modo de realización puede ser encontrado en el manual de la referencia [11]), es configurar la red. Ésta ha de hacerse obligatoriamente a través de línea de comandos, lo que requirió de cierto tiempo para ser descubierto. Esto es así ya que, por falta de actualizaciones y corrección de versiones, el kernel presenta ciertos fallos y no integra totalmente la interfaz.

Para configurar la red por medio de terminal se necesitan ejecutar los siguientes comandos:

- `nano /etc/network/interfaces en etho configuramos la ip, subred y puerta de enlace`
- `nano /etc/resolv.conf  en nameserver ponemos el dns`
- `sudo /etc/init.d/networking restart Reiniciamos la red.`

Nano es un editor de texto “en línea”, instalado por defecto en todo sistema Linux, que permite modificar desde el terminal los ficheros abiertos en modo texto.

### 5.5.2. Puesta en marcha y uso básico del robot Shadow Hand.

#### *5.5.2.1. Secuencia de Arranque.*

La secuencia a seguir para encender y usar la mano robótica Shadow Hand es la siguiente:

a) Conectar la alimentación.
b) Esperar unos segundos para estabilizar la alimentación.
c) Conectar el cable CAN.
*d)* Clickar sobre el botón de una esfera verde denominado “*Start the Robot”.*
*e)* Inicializar los controladores con la utilidad “*Start Position Controllers*”, cuya imagen es la de una especie de esfera azul.
*f)* Lanzar la aplicación *Joint Sliders* y comprobar que los dedos se mueven correctamente.

Todas las utilidades y aplicaciones mencionadas se encuentran en el escritorio.

#### *5.5.2.2. Secuencia de Apagado.*

Antes de comenzar es conveniente asegurarse de que todos los procesos han finalizado y no existe ninguna utilidad abierta, relacionada con el robot. Una vez hechas estas comprobaciones, el procedimiento a seguir es:

a) Detener los controladores mediante la utilidad que asemeja un circulo grisáceo denominada *“Stop All Cotrollers”.*
b) Detener el robot por medio de la utilidad *“Stop the Robot”,* anexa a la anterior en cuanto a situación.
c) Desconectar el cable de CAN.
d) Desconectar la alimentación.

NOTA: Recordar que la mano ha de situarse en la posición en la que se quiera guardar antes de iniciar el proceso.

#### *5.5.2.3. Relación y descripción de utilidades instaladas.*

A continuación se enumera y define brevemente el conjunto de utilidades que puede encontrarse en el ordenador con Debian. Todas ellas accesibles desde el escritorio principal.

- **Start the Robot.** Inicia la comunicación por CAN y algunos procesos básicos de configuración.
- **Start Position Controllers.** Inicia todos los procesos necesarios para controlar el motor, incluyendo la adquisición de señales y envío de órdenes. Debe ejecutarse tras la anterior.
- **Stop All Controllers.** Operación antagónica a la anterior.
- **Stop the Robot.** Detiene todos los procesos relacionados con el robot. Debe ser ejecutada una vez detenidos los controladores.
- **Robot tree.** Muestra información relativa a las comunicaciones disponibles y estado del hardware.
- **Terminator.** Llamada de parada de emergencia.
- **Motor Status.** Estado de los motores. Empleada principalmente para determinar si algún motor se encuentra dañado o en zona de riesgo.
- **ShowValuesPalm.** Muestra, en tiempo real, los valores de todos los actuadores y sensores de la mano.
- **Joint Sliders.** Permite desplazar todas las articulaciones sin necesidad de programar. Muy útil para probar funcionalidades y operaciones de debug básicas.
- **H-Bridge.** Permite configurar todos los aspectos necesarios para controlar el robot de forma directa por señales externas. Esta utilidad forma parte de las modificaciones exclusivas de este robot.

#### *5.5.2.4. API's y Librerías auxiliares empleadas para el control.*

En los manuales de API's y Software desarrollado [12] [13] se puede encontrar una gran variedad de funciones así como procedimientos para modificar distintos procesos: calibración, ajuste de los PIDs de control utilizados, datos manejados por las funciones…

Los programas del robot se pueden realizar en c o c++, ya que las librerías son de c clásico. En este apartado se enumerarán las librerías a enlazar en los MakeFiles, principales librerías a modificar y todas las funciones utilizadas para desarrollar las aplicaciones citadas en el *Capítulo 6. Software Desarrollado.*

**Funciones usadas en la comunicación con el robot.**

Para leer y enviar datos al robot se han empleado principalmente cuatro funciones y un tipo de estructura específico:

- ***robot_read_sensor(&dato_tipo_sensor).*** Lee de un sensor pasado como parámetro y devuelve un valor float con la medición.
- ***robot_init().*** Comprueba si el robot ha sido iniciado. En caso de fallo devuelve -1.
- ***robot_name_to_sensor("FFJ3_Target",&dato_tipo_sensor).*** Almacena los datos del "sensor" pasado como primer parámetro en una variable de tipo sensor correspondiente al segundo parámetro.
- ***robot_sensor_update(&dato_tipo_sensor, 0.0).*** Mueve el actuador pasado como primer parámetro a la posición marcada por el segundo parámetro.
- ***struct sensor***. Estructura específica para trabajar con la mano.

**Librerías a incluir en el archivo cpp.**

Para poder usar las funciones y tipos de datos desarrollados por Shadow es necesario incluir las siguientes librerías en el archivo fuente:

```
#include <robot/robot.h>

#include "RobotConfig.h"
```

Como se puede comprobar la librería RobotConfig deberá copiarse al directorio desde el que se lance el MakeFile de compilación. En cuanto al MakeFile.txt, es necesario realizar un enlace del código fuente al paquete de librería */usr/lib/robot.a* así como especificar que se incluyan, mediante la directiva include de precompilación, los directorios realtime y kernel de Linux a las librerías desarrolladas. Se puede encontrar un ejemplo de MakeFile en el cd adjunto.

**Principales librerías a modificar.**

Si se desea modificar algún parámetro relativo a los "sensores" (recordar que Shadow se refiere con este término a todas las entradas y salidas) se deberá revisar, en primer lugar, las librerías especificadas a continuación. Para encontrarlas, la forma más cómoda es acudir a una ventana de navegación y, seleccionando *file system* como rango de búsqueda, proceder a realizar una búsqueda del archivo en todo el sistema de archivos.

- **rtio sensor.c.** Posee información sobre todas las librerías relacionadas con los sensores.
- **robot/sensor.h.** Contiene información sobre todas las funciones y tipos de datos relacionados con los sensores.
- **robot/robot.h.** Cabecera de las funciones y tipos de datos del resto de parámetros del robot. Mediante su consulta es posible deducir muchos erros y fallos.

## 5.6. Relación de Problemas Encontrados y posibles soluciones.

A continuación, se citarán los principales problemas encontrados durante la instalación y manejo del software mencionado anteriormente, algunos de los cuales supusieron grandes lastres hasta que se encontró una solución:

| Problema | Posible Solución |
|---|---|
| Pantalla negra en Ubuntu después del grub. | Debido a fallos del controlador gráfico. Conectar un monitor externo y comprobar si lo reconoce. Si no es posible, acceder en modo terminal como superusuario y deshabilitar el controlador. |
| Pantalla negra en Ubuntu con letras amarillas tras la carga del grub. | Problema del controlador ATI AMD Catalyst. Desinstalarlo y usar FGLRX, privativo – opción de lanzamiento. |
| El trackpad de mac no funciona. | Ajustar la calibración en el menú de ajustes, concretamente aumentar la sensibilidad. |

| | |
|---|---|
| Ubuntu no inicia el grub. | Probablemente el grub no haya sido instalado o la partición se ha creado en último lugar, tratar de marcarla como preferida de arranque. |
| Kernel Failure | Restablecer a una versión de kernel anterior desde el grub. |
| Al instalar Ubuntu no se puede volver a acceder al otro sistema instalado. | Este problema está relacionado con las opciones de arranque rápido de las nuevas BIOS que se encuentran en el mercado. |
| Simulador de Gazebo de Shadow aparece en posición inicial y no publica en ningún topic. | Problema durante la descarga de paquetes, este problema afecta únicamente a Groovy. |
| Simulador de Gazebo de Shadow aparece con problemas de consistencia y estabilidad, quedando el robot contenido en el interior del soporte verde. | Problema que presenta el simulador a día de hoy y que debe ser resuelto por Shadow. |
| Al arrancar ROS (roscore) muestra un mensaje señalando que las dependencias estaban configuradas antes para otra versión. | Esto se debe a un cruce de dependencias ocasionado por una mala instalación de dos distribuciones de ROS. Modificar el fichero de carga de dependencias (bashrc) para configurar la versión a usar. |
| Al usar Groovy no funciona PCL. | Instalar la versión 1.7 de PCL y marcar en los CmakeLists que se usé como mínimo esta versión para compilar. |
| El robot de Shadow no arranca correctamente, señalando que no encuentra los controladores. | Si se ha seguido el proceso señalado en el anexo específico, el problema se encuentra en el cable de comunicación CAN o algún punto del trazado de la comunicación. |
| En Debian, salta un mensaje de error, alegando: FATAL KERNEL FAILURE. | La versión presenta problemas de estabilidad, guardar los archivos de interés y reiniciar. |

# CAPÍTULO 6.
# SOFTWARE DESARROLLADO.

## 6.1. Introducción.

A lo largo del presente capítulo se citarán todos los desarrollos software realizados que permiten satisfacer los criterios y objetivos señalados en el apartado *1.3 Objetivos* del capítulo 1. Dada su amplitud y conjunto de campos abarcados, el proyecto puede quedar subdivido en tres fases o etapas de desarrollo, a saber:

- Control de un manipulador antropomórfico robótico para lograr agarres estables y precisos.
- Desarrollo de un sistema de Visión Artificial.
- Sincronización de dos Robots.

Puesto que cada una de estas etapas presenta dificultad propia y visto el grado de problemas que han debido afrontarse, se ha considerado organizar el conjunto de desarrollos realizados atendiendo a la etapa de pertenencia. Así mismo, dentro de cada fase de desarrollo, se mostrarán los programas realizados en orden de dificultad creciente, señalándose los problemas encontrados y estrategias desplegadas.

Por último, destacar que este trabajo sólo mostrará fragmentos del código completo que sean considerados relevantes o aclaratorios. Pudiendo encontrarse las cabeceras y programas aquí mencionados en el CD adjunto.

## 6.2. Control de un manipulador antropomórfico robótico para lograr agarres estables y precisos.

Este proyecto supuso la primera toma de contacto con un manipulador robótico de precisión. Por ello, ha sido desarrollado un amplio conjunto de programas que permiten facilitar el aprendizaje y comparar distintas técnicas y estrategias de movimiento para lograr un agarre estable.

Una vez presentadas y analizadas estas opciones de movimiento se entrará a debatir la temática inherente al control de la presión ejercida y medidas antideslizamiento.

Por último, se dará por concluido este apartado citando las opciones de integración con ROS del robot y justificando la opción escogida.

### 6.2.1. Primeros programas: análisis del movimiento.

Dejando de lado todos los desarrollos que involucran movimientos sencillos, como puede ser el desplazamiento secuencial de las articulaciones proximales con gesto desesperado, el primer programa de interés realizado consistió en tratar de mover todas las articulaciones un mismo ángulo, con objeto de comprobar la eficiencia de desplazamiento y capacidad del robot para alcanzar una pose.

El programa denominado **cerrar_mano** tiene como objetivo primordial desplazar todas las articulaciones en un intento de definir una forma de garra en la que las yemas de los dedos se encuentran en oposición al pulgar. Los desplazamientos, análogos para todas las articulaciones físicas de flexión, se determinaron de forma tal que los dedos no entrasen en contacto ya que los sistemas de presión y seguridad aún no habían sido desarrollados en su plenitud.

En contraste con lo esperado, la prueba de este código reveló ciertos problemas de índole mecánica en el robot. Al tratar de abarcar ángulos de movimiento superiores a los 20º, los dedos realizaban inducciones entre ellos. Este fenómeno provoca que; una vez un dedo alcanza su posición final, ésta puede ser alterada temporalmente mientras otro dedo se posiciona. Señalar que dichas inducciones se producen principalmente entre los dedos índice y anular.

Recordar que las API's presentan usos propios de funcionalidades de tiempo real, esto quiere decir que es desaconsejado acudir a estándares de C++ tipo POSIX, ya que implicaría manejar dos módulos de tiempo real. Así, evitando caer en redundancias de tiempo real que pueden ocasionar problemas de comunicación y siguiendo las recomendaciones de la empresa Shadow, se decidió abordar la resolución del problema desde un punto de vista de movimiento secuencial que permita "mostrar" un comportamiento paralelo. Son muchas las estrategias de programación secuencial existentes que tratan de simular comportamientos de tiempo real. Puesto que, para esta aplicación, un retraso de milésimas de segundo puede ser considerado despreciable se optó por encauzar la estrategia hacia el uso de iteraciones con anidación de tipo for.

No obstante, a pesar de este mínimo retardo, las inducciones seguían existiendo. De este modo, el siguiente paso se centró en determinar en qué situaciones se producían las inducciones y cuales garantizaban la no aparición de éstas. Tras varios experimentos se comprobó que los ángulos inferiores a 20º presentaban una tasa de inducciones inferior al 10%. Además, éstas disminuían exponencialmente conforme el ángulo desplazado se aproximaba a los 10º. Se concluyó así en realizar un código que partiese o trocease las trayectorias a realizar en pequeñas partes que implicasen un desplazamiento de 10º.

De esta forma, se decidió que el nuevo código desarrollado recibiría la trayectoria final, mediante defines, y la procesaría para provocar un movimiento articular similar al de un motor paso a paso. Sin embargo, este patrón secuencial de movimientos introducía nuevos retardos para alcanzar la posición final dada.

Los programas **cerrar_mano_dedo_serie** y **cerrar_mano_dedo_art_part** son los primeros desarrollos que incluyen esta nueva estrategia y tratan de dar solución a los retardos que introduce.

Por un lado, **cerrar_mano_dedo_serie** desplaza los dedos secuencialmente, esto es, las metas articulares se alcanzan para cada dedo lo que provoca que primero se posicione el índice, posteriormente el anular y por último el pulgar.

Por otro lado, **cerrar_mano_dedo_art_par** presenta una estrategia opuesta: las posiciones finales se alcanzan por orden articular, esto es, primero se posicionan todas las proximales, posteriormente las medias y así sucesivamente.

Aunque ambos programas consiguen eliminar las inducciones, el movimiento por orden articular permite agarres más estables. El movimiento que consigue el segundo programa provoca que el contacto entre las yemas de los dedos y el objeto se produzca de forma "simultanea", mientras que el contacto del programa **cerrar_mano_serie** es de tipo secuencial (transcurre más tiempo entre los contactos de las yemas) lo que favorece el desequilibrio y compromete la estabilidad.

Una vez definida la pauta de movimiento, se comenzó a diseñar el programa final que será analizado en el siguiente apartado.

### 6.2.2. Programa Final para el Agarre.

El programa final se ha desarrollado en C++ como si de una versión comercial de código se tratase. De este modo, no sólo se han desarrollado librerías propias de control sino que se hace uso de la declaración de clases. Se ha creado así una clase denominada *usarMano* que aglutina el conjunto de funciones y variables necesarias para el control, distinguiendo entre miembros private, protected y public acorde a los principios de buena programación aprendidos a lo largo del grado cursado y de consultar el libro [22] . A continuación se analizarán los distintos procesos de interés que tienen lugar en el programa final.

#### *6.2.2.1. Adquisición de datos.*

En el código final, las dimensiones del objeto a coger son leídas desde un fichero. Este fichero actúa como una base de datos de distintos objetos, especificando el usuario, al lanzar el programa, el objeto a coger como argumento.

Actualmente se contemplan tres objetos:

- Botella. Establecidas las dimensiones de una botella de agua de 1.5L con diámetro próximo a 9 cm.

[22 ] Cómo Programar C++. P.J.Deitel & H.M.Deitel. Sexta Edición. Ed. Pearson-Prentince Hall.

- Caja. Consta de unas dimensiones de 26,6 cm de altura, 6,4 cm de fondo y 14.8 cm de ancho. Como hándicap para el agarre su superficie es deslizante lo que limita las opciones de agarre.
- Ambientador. Bote de ambientador medio lleno de 45cm de altura y 7 cm de diámetro. Este objeto se usa como elemento de peso variable.

Este fichero de objetos es reutilizado para obtener las dimensiones de la mano, concretamente la longitud de las falanges. A continuación, se puede observar un ejemplo de introducción de datos:

```
Características dimensionales del objeto 1 (índice =0) (ver funciÃ³n)
[distancia centro masa, radio objeto, long falange 1, long falange 2]
(parámetros botella rosa)
4.45
4.445
4.5
2.5
```

La distancia al centro de masa se define como la longitud medida desde la parte superior del metacarpo hasta el centro del objeto. En caso de que esta distancia sea inferior o igual al radio del objeto, el algoritmo cinemático programado fallará y el programa finalizará con un mensaje de error. Las ecuaciones que componen el algoritmo cinemático desarrollado pueden encontrarse en el *Anexo 9. Algoritmo cinemático programado.*

#### *6.2.2.2. Control de presión.*

Este proceso comienza leyendo los valores de presión que supuestamente permitirían un agarre estable, para ello lee del fichero valores_presion_estable.txt. Como se pudo comprobar de varios experimentos, los valores de presión captados por las galgas en situaciones estándar (mano encendida y desplegada) son variables. Para más información sobre este punto consultar el *Anexo 10. Variabilidad de presión en los sensores de la Shadow Hand.*

Una vez leídos los valores de presión a alcanzar, se produce un movimiento paralelo análogo al de posicionamiento que, en lugar de concluir con la posición final a alcanzar, finaliza cuando los sensores de presión marcan una presión igual a la leída con una tolerancia de 0.1 bar (ruido de los sensores),especificada por la

variable `TOLERANCIA_PRESION`. Así, si la presión es superior a la marcada como estable o referencia, los dedos se alejaran, y si es inferior se aproximarán. Además, se considera que este desplazamiento será de aproximación o precisión por lo que si un dedo presenta, por ejemplo, un déficit de presión, todas sus articulaciones incrementarán su posición en 2,5º tal y como marca la variable `INCREMENTO_CORRECCION_PRESION`. Este valor, que puede parecer excesivo para una aproximación, es el mínimo necesario para producir un cambio en la posición de los dedos, debido a la zona muerta de los motores.

```
    for( int z=0;z<numero_dedos;z++) leer_sensor(z);


    for( int z=0;z<numero_dedos;z++){ //se desplazarán los tres dedos pero incrementalmente.

        bool agarre_estable=false;

        //Mientras no sea estable, comprobar si es estable. En caso de que no lo sea, si la presión es superior a la debida, alejar. Si es inferior, apr

        while (!agarre_estable && !problema){
            if(sensores[z].presion>=sensores[z].pres_estable-TOLERANCIA_PRESION && sensores[z].presion<=sensores[z].pres_estable+TOLERANCIA_PRESION) aga

            else if (sensores[z].presion>sensores[z].pres_estable || sensores[z].presion<sensores[z].pres_estable ) {

                for(int i=0; i<numero_articulaciones; i++){ //análogo al paso a paso (MOV_PASO)

                    if( ((z==0 || z==1) && (i==1)) ||(z==2 && (i==2||i==3))||i==2));
                    else{

                            leer_sensor(z,i);
                            if (sensores[z].presion>sensores[z].pres_estable)
                                desplazamientoRelativo=sensores[z].posicion_art[i]- INCREMENTO_CORRECCION_PRESION;

                            else if (sensores[z].presion<sensores[z].pres_estable)
                                desplazamientoRelativo=sensores[z].posicion_art[i]+INCREMENTO_CORRECCION_PRESION;

                            if( movimiento_posible(z,i,desplazamientoRelativo)) control_actuador(z,i,desplazamientoRelativo);
                            else {
                                problema=true;
                                cout << "\n LOGRAR EL AGARRE ESTABLE IMPLICA EL DESPLAZAMIENTO DE UN DEDO A POSICIONES SINGULARES."<< endl;
                                cout << " SITUACION ANOMALA AL TRATAR DE REALIZAR EL AGARRE. ABORTANDO EJECUCION"<< endl;
                                return false;
                            }


                    }
                }
            }


            if(!agarre_estable) {
                leer_sensor(z);
                cout << "\n La presion de "<< nombres[z].presionID<< " tiene un valor de "<< sensores[z].presion << endl;
            }
        } //fin while agarre estable
    } //fin for de agarre
    return true;
}
```

*Ilustración 29. Fragmento del bucle de ajuste de presión.*

### *6.2.2.3. Control Antideslizamiento.*

Este segundo control trata de corregir la posición de los dedos para afrontar aquellos casos en los que la presión es insuficiente, ya sea por fallos de acondicionamiento o incremento de peso inesperado en el objeto (la botella se ha usado siempre vacía, podría llenarse).

El movimiento sigue basándose en la arquitectura paralela ya comentada. Experimentalmente se comprobó, como puede verse en las imágenes, que al tratar de deslizar el objeto agarrado los valores de presión varían un valor superior a 0.3 bares , positivos o negativos según la dirección del esfuerzo. Así, para la realización del algoritmo, se parte de la hipótesis de que toda variación superior a este valor implica deslizamiento, ya que se considera el objeto irrompible pues no se realizan agarres precisos ni trabaja con objetos frágiles.

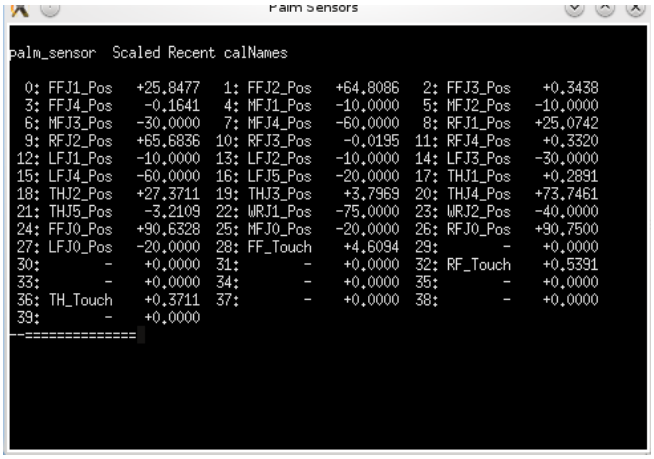



*Ilustración 30. Valores de los sensores agarrando una botella.*

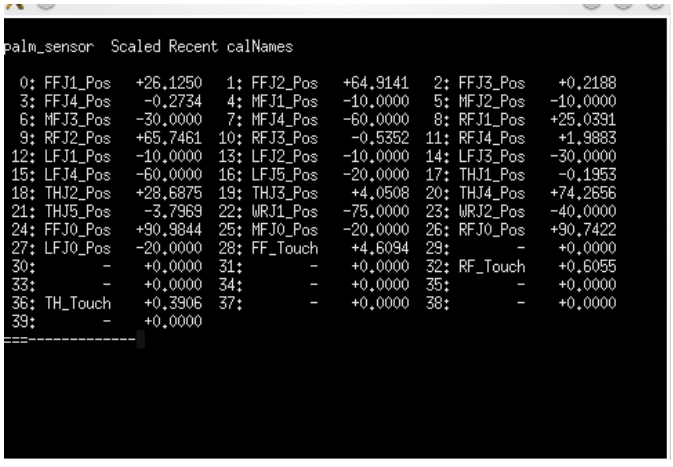



*Ilustración 31. Valores de los sensores cuando la botella que agarra desliza por acción de una esfuerza externa.*

Para ello, el robot entra en un bucle que puede programarse infinito o marcarse una pauta de salida, por defecto se usa este último. Desde el punto de vista finito, existe una variable denominada `CONTEO_ANTI_DESLIZAMIENTO` que determina el número de veces consecutivas, en las que no se ha detectado indicio de deslizamiento, que definen una situación segura. De este modo, el robot lee los valores de presión de los sensores en un determinado instante, espera un cuarto de segundo y vuelve a leerlos. Si la diferencia entre ambos valores supera una tolerancia de ruido, definida por `TOLERANCIA_DESLIZAMIENTO`, el robot considera que el objeto está deslizando por lo que aumenta la presión del agarre, incrementado la posición de todas las articulaciones en 1º, valor marcado por `INCREMENTO_CORRECCION_DESLIZAMIENTO.`

Se observa que, en este caso, a pesar de lo comentado, el valor incrementado se encuentra dentro de la zona de deslizamiento, lo que provocará que en ocasiones no se desplace el robot. Esta decisión se ha tomado debido a la estructura del bucle realizado: la condición de salida es opuesta al caso que ocurre por defecto (si el objeto desliza y no se mueve el robot, seguirá deslizando por lo que el bucle no terminará) y, en caso de no desplazarse, no se compromete la estructura del robot (caso contrario a un ajuste de presión por exceso).

```
754         // medida de la presion inicial
755         cout << "\n Presion Medida:\n "<<endl;
756 
757         // medida de la presión ejercida
758         for (int i =0; i< numero_dedos; i++){
759             presionAnterior[i]=sensores[i].presion;
760             cout <<" "<< nombres[i].presionID<<" -->"<< presionAnterior[i]<<endl;
761         }
762         usleep (250000);
763 
764         // tras esperar un tiempo, se vuelve a medir la presión para determinar si hay variación. En ese caso, se activa la baliza y procede al movimiento de los dedos.
765         cout << "\n Presion Medida tras 1/4 segundo:\n "<<endl;
766 
767         for (int i=0; i<numero_dedos;i++){
768 
769             if(sensores[i].presion!= -10){
770 
771                 leer_sensor(i);
772                 cout <<" "<< nombres[i].presionID<<" -->"<< sensores[i].presion<<endl;
773 
774 
775                     if( sensores[i].presion<=presionAnterior[i]-TOLERANCIA_DESLIZAMIENTO || sensores[i].presion>=presionAnterior[i]+TOLERANCIA_DESLIZAMIENTO){
776 
777                         if( sensores[i].presion < LIMITE_SEGURIDAD_PRESION) {
778                             diferenciaPresion=true;
779                             for(int j=0; j < numero_articulaciones; j++)
780                                 if( ((i==0 || i==1) && (j==1||j==2)) ||(i==2 && (j==2||j==3))); //if para evitar mover articulaciones no deseadas.
781                                 else{
782                                     leer_sensor(i,j);
783                                     desplazamientoRelativo=sensores[i].posicion_art[j]+ INCREMENTO_CORRECCION_DESLIZAMIENTO;
784                                     if( movimiento_posible(i,j,desplazamientoRelativo)) control_actuador(i,j,desplazamientoRelativo);
785                                     else sensores[i].presion=-10;
786                                 }
787                         } else {
788                             cout << "\n PRESION EXCEDIDA PARA EL DEDO"<< nombres[i].presionID << endl;
789                             problema=true;
790                         }
791 
792                     } // if comprobar diferencia presion
793                 } else {
794 
795                     cout << " EVITAR EL DESLIZAMIENTO IMPLICA EL DESPLAZAMIENTO DE UN DEDO A POSICIONES SINGULARES."<< endl;
796                     problema=true;
797                 }
798 
799             }
800 
801 
802         } while( iteraciones< CONTEO_ANTI_DESLIZAMIENTO || problema);
803 
804         if (!problema) return true;
805         else
806             cout << " SITUACION ANOMALA AL TRATAR DE EVITAR DESLIZAMIENTOS. ABORTANDO EJECUCION"<< endl;
807             return false;
808 
809 
```

*Ilustración 32. Extracto del bucle antideslizamiento.*

#### *6.2.2.4. Funciones Auxiliares y seguridad.*

Además del funcionamiento básico presentado, la cantidad de bucles y movimientos automáticos recursivos programados exige establecer medidas de seguridad que eviten daños al equipo.

La primera verificación se encuentra en el objeto a coger, en caso de no existir éste en la base de datos, la función **decodificar_objeto**() manda un mensaje de error.

Las siguientes comprobaciones que se realizan se localizan en la lectura de los ficheros, en caso de leerse valores ilógicos se procede a la finalización del programa. La función encargada de ejecutar el KDL programado también realiza comprobaciones de valores alcanzables por el robot, esta función además integra subllamadas a funciones de mapeado que adaptan los valores calculados a valores equivalentes en el rango de movimiento de los dedos.

La primera seguridad de desplazamiento se encuentra en los propios bucles de movimiento. Cada una de las tres funciones que involucran movimiento (posicionamiento, control de presión y control antideslizamiento) contempla una estructura if, como se ha podido observar, que impide que las articulaciones que no deben moverse accedan a sentencias de movimiento. Además, una vez finalizada la función de posicionamiento (colocar las articulaciones en la posición final determinada por el algoritmo cinemático) se entra en un estado de comprobación del movimiento. Si la diferencia entre la posición final y la calculada superan un margen de tolerancia, definido por TOLERANCIA_POSICION (por defecto en 0.2 º), el programa se detendrá señalando la existencia de problemas en los motores de la articulación problemática. Con esta última comprobación hay que tener cuidado ya que la velocidad de procesado del ordenador es claramente superior a la de posicionado físico de las articulaciones, por lo que es necesario contar con un tiempo de espera suficiente para este “retardo físico”.

En segundo lugar, antes de mover cualquier articulación se realiza una llama a la función auxiliar **movimiento_posible()** que determina no sólo si la articulación goza de privilegios de movimiento, sino que también evalúa si el movimiento solicitado es alcanzable.

Para la evaluación del movimiento alcanzable recurre a un conjunto de defines cuyos valores por defecto son:

```
 // EN CASO DE ALCANZARSE ALGUNO DE ESTOS VALORES SE SUSPENDE LA
EJECUCION.
#define RANGO_SEGURO_ART_0 130
#define RANGO_SEGURO_MAPEADOS_PULGAR 30
#define RANGO_SEGURO_ART_3_DEDOS 90
#define RANGO_SEGURO_ART_1_PULGAR 90
```

Como se puede observar, todos los valores se encuentran próximos a sus máximos salvo los de la articulación virtual 0, esta limitación supone una seguridad de gran importancia totalmente dependiente del hardware. Los sensores de presión usados por Shadow están constituidos por galgas extensiométricas, colocadas de tal manera que la precisión en el centro de las yemas es máxima. No obstante, la sensibilidad decae exponencialmente conforme nos aproximamos a los extremos o límites de las yemas, siendo el robot insensible a los objetos que sostiene con la punta o laterales de los dedos. Este hecho que puede parecer irrelevante o fácilmente salvable, supone un gran problema cuando se trata de realizar un ajuste de presión o evitar un deslizamiento.

El principal problema deriva de la imposición de la empresa para mover las articulaciones distales y medias de los dedos conjuntamente. Se tiene que, al realizar un ajuste de presión, dado que la posición es la adecuada, el aumento de sujeción debería lograrse mediante el movimiento de la articulación media de modo tal que la posición de la yema respecto al objeto no sufriese modificaciones. La articulación virtual 0 impide este tipo de movimiento, obligando a realizar un desplazamiento conjunto y aleatorio, por ello existen situaciones en las que los dedos al tratar de evitar un deslizamiento entran en zonas de sensibilidad muerta que, de no ser evitados, finalizan con una flexión total. Puesto que no es posible recuperarse de estas situaciones con un agarre estable, se ha decidido programar una salida prematura cuando se alcanza una de estas zonas ($130^{o}$ es la curvatura máxima requerida para sujetar cualquiera de los objetos de estudio en el peor de los casos).

Por último, en varios puntos del código se comprueban los valores de presión y, en caso de excederse los 10 ó 2 bares (según el dedo), se aborta la ejecución.

#### *6.2.2.5. Desglose del programa.*

Aunque hasta el momento sólo se ha hablado de un programa, el desarrollo completo consta de dos programas: **Ajuste_Presion** y **Agarre_Presion**. El primero de ellos, sirve de utilidad, con interfaz de usuario sencilla, para determinar la presión de referencia que debe marcarse para agarrar un objeto. Por ello, este programa forma parte de la fase de configuración del desarrollo, a veces denominada etapa de entrenamiento. En la siguiente imagen se muestra la interfaz que aparece una vez se ha alcanzado la posición final calculada por el algoritmo cinemático, sin entrar en procesos de control de presión.


*Ilustración 33. Ejemplo de uso del programa de "configuración" Ajuste_Presion.*

A pesar de su simpleza, todas las medidas de seguridad antes mentadas han sido integradas.

Por otro lado, el programa Agarre_Presion constituye el programa con el funcionamiento global cuyas funcionalidades ya han sido analizadas a lo largo de los apartados anteriores. Únicamente señalar que éste cuenta con un modo *debug* en el que es el usuario el que introduce todos los valores, haciéndose pasar por el robot, lo que permite prevenir gran cantidad de fallos.

#### *6.2.2.6. Opciones de integración en ROS.*

Dado que uno de los objetivos del proyecto es adaptar el programa presentado para sincronizarse con un brazo robótico, programado en ROS, y recibir la información del objeto a coger desde un sistema de visión artificial, integrado también en ROS, se debieron evaluar diversas opciones de integración:

**Integración de todo el proyecto en ROS y usar Debian como intérprete.**

La primera opción planteada fue trabajar directamente en ROS transcribiendo todo el código para adaptarlo a ROS. Esta tarea que puede resultar bastante compleja y problemática, no tuvo mayor importancia ya que desde un principio se tuvo en mente la posibilidad, por lo que con tan solo cambiar cinco líneas de código el programa queda integrado en su totalidad en el nuevo sistema.

Para las pruebas de esta opción se empleó el simulador de gazebo con los modelos proporcionados con Shadow. Además, con ayuda de Toni Oliver y Armando de La Rosa (ambos desarrolladores de Shadow), se creó un paquete que permitía simular el conjunto de la mano y el sistema de visión artificial. Para más información sobre la construcción del simulador y creación de los paquetes necesarios, consultar el *Anexo 11. Proceso de Desarrollo de un simulador del robot junto a Kinect.*

Aunque la migración fue satisfactoria y no se produjo ningún problema a nivel de programación, la cantidad de sockets a abrir y sincronizar era tal que las posibilidades de errores no eran aceptables. La paradoja de este proceso se encuentra en la propia definición de ROS, como middleware éste trata de hacer invisible la comunicación para el usuario. Al tratar de usar sockets propios es necesarios realizar envolturas de código. Puesto que, de trabajos anteriores (integración de un robot móvil con Arduino en ROS), ya se conocían los problemas que esto ocasionaba dada la redundancia de comunicaciones generada e imposiciones de refresco marcadas por el middleware, se decidió evitar mantener y gestionar 29 nodos de ROS para satisfacer las comunicaciones, 16 dedicados a sensores y 13 a actuadores. A pesar de todo, en el CD se puede encontrar una batería de 10 programas desarrollados para controlar el robot Shadow Hand en ROS.

**Uso de Moveit!**

Una segunda opción contemplaba el uso de Moveit! , no obstante ésta fue rápidamente desechada ya que:

El kdl de moveit!, como se explicó en la ROS-RM, no es capaz de trabajar con robots de menos de 6 grados de libertad, aunque con una programación muy precisa puede lograrse el control de robot de 5 GDL, tal y como señaló Sachin Chitta. Así, la solución pasaría por desarrollar plugins propios, siguiendo un procedimiento similar al que mostró el conferenciante Guillaume Walk, experto en manipulación dinámica precisa, en el workshop ROS-RM.

Además, dejando de lado el nivel de conocimientos exigido para desarrollar un kdl propio, el problema de la comunicación multisocket sigue existiendo.

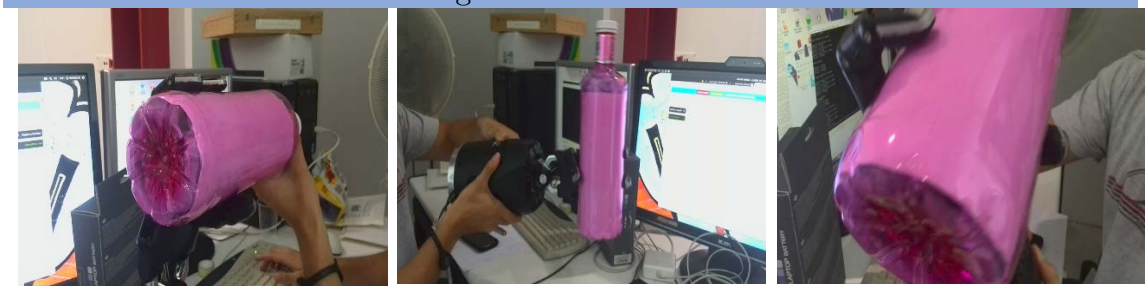
**Uso de sistemas independientes y comunicación por socket.**

Ésta ha sido la opción adoptada, dada la comodidad y eficiencia mostrada. Dicha estrategia se basa en mantener ROS con el brazo y el sistema de visión aislado y, en otro equipo, el control de la mano en Debian. Para implementar esta solución ha sido necesario desarrollar un conjunto de librerías propias de comunicación con funciones de paso de tokens, pings y protocolos de verificación de envío.

### *6.2.2.7. Demostración de Agarres.*

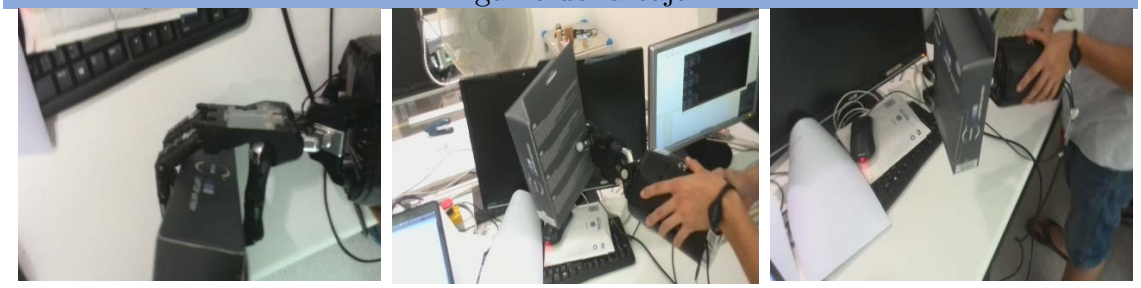
A continuación se mostrará un conjunto de fotos en las que se puede comprobar la validez de los agarres con distintos ángulos e inclinaciones del objeto.

Agarre de la botella



Agarre del ambientador

Agarre de la caja



De todos los agarres realizados, el más estable y seguro fue el del ambientador, a pesar de su peso variable, con una probabilidad de acierto del 90%, seguido por la botella con una probabilidad del 80% (siendo uno de los fallos debido a atranque del pulgar y el segundo detenido por alcanzar zonas insensibles). La caja, dada su superficie deslizante y ángulo que debe formar el pulgar obtuvo una probabilidad de acierto del 60% (dos fallos de agarre y dos por parálisis del pulgar. Para la obtención de estas estadísticas se realizaron 10 pruebas de *grasping* por objeto.

### 6.2.3. Problemas encontrados en el manipulador.

A lo largo del proyecto han sido numerosos los problemas, de diversa índole, que han debido de ser salvados. Entre ellos, destacan los problemas relacionados con el hardware del robot manipulador *Shadow Hand C6M2 modificado*, recogiéndose en la siguiente tabla, junto a la solución abordada, los más importantes:

| Problema | Solución Implementada. |
|---|---|
| Los sensores de presión no se encuentran calibrados de forma equivalente ya sea por motivos de acondicionamiento o problemas relacionados con la sensorización. Concretamente, el dedo índice presenta una sensibilidad 10 veces superior al resto. | La principal solución desarrollada ha sido tratamiento a nivel software de estas diferencias. |
| Las galgas extensiométricas de los sensores se encuentran colocadas de forma tal que el robot carece de sensibilidad suficiente para realizar un control de presión adecuado si el punto de contacto no se localiza en el centro de las yemas. | Modificación de los algoritmos e inclusión de procedimientos de salida. Situación insalvable. |
| El valor entregado por los sensores, en condiciones estándar (mano abierta y sin entrar en contacto) varían en cada encendido. | Trabajar con valores de presión relativa y mejora de los controles de presión y antideslizamiento. |
| La articulación 2 del segundo dedo se atasca esporádicamente, siendo necesario mover el resto de articulaciones para liberar el tendón. | No corregible de forma total a nivel software. Verificar las posiciones a alcanzar y adaptar los algoritmos para reducir el número de desplazamientos de la articulación problemática. |
| Se presentan inercias entre los dedos, a excepción del pulgar, que provocan que el movimiento de uno de ellos induzca el movimiento del otro. Sólo acontece para desplazamientos amplios | Programación de movimientos paso a paso que dividan la trayectoria en pequeños desplazamientos. |

## 6.3. Visión Artificial: PCL y cámara de profundidad.

El código del sistema de visión se divide en 4 archivos. Uno principal que actúa como maestro al encargarse de crear los ficheros que guardarán la información obtenida por el reconocimiento de la cámara, llamar al programa de reconocimiento 3D y al de control del brazo. Todo ello a través de una interfaz de usuario que permite elegir el objeto que se quiere coger de entre los que se han detectado.

El segundo programa es el encargado de obtener las imágenes de la cámara. Con él se consigue la información necesaria para crear una pequeña base de datos de objetos y escenas con las que posteriormente se podrá trabajar. Este programa se ha creado en base al código ofrecido en el siguiente enlace, el cual ha sido ligeramente modificado para adecuarse a los fines del presente proyecto.

http://robotica.unileon.es/mediawiki/index.php/PCL/OpenNI_tutorial_1:_Installing_and_testing

El tercer programa es el que se encarga de extraer de una escena cualquiera los objetos que se encuentran depositados sobre la mesa. Es capaz de discretizar la información y obtener los clúster individuales que conformarán los modelos que servirán para poder identificar los objetos que se encuentren en la escena experimento.

El último programa es el que realmente hace el reconocimiento de objetos. Este programa trabaja con modelos y escenas almacenadas en unas carpetas específicas y acaba escribiendo en un fichero las coordenadas del centroide de los objetos detectados en la escena.

A continuación se presenta un flujograma simplificado del modo de operación del sistema de visión que se ha diseñado:

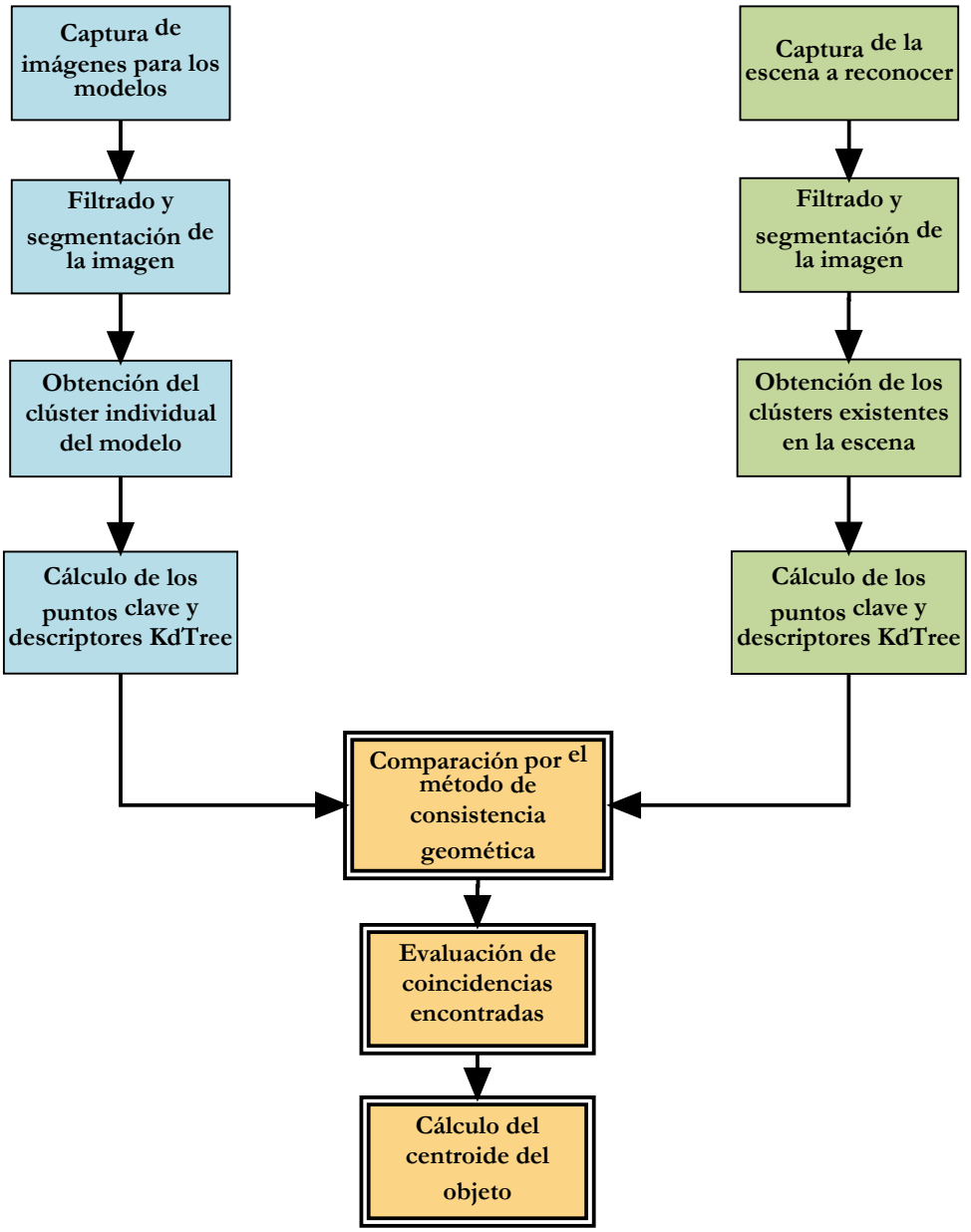



*Ilustración 34.Flujograma del programa de reconocimiento de objetos 3D.*

### 6.3.1. Programa Principal.

El programa principal se vale de una estructura (`objetos_conocidos`) que se ha diseñado con el fin de identificar los objetos para poder organizar las llamadas a los otros programas que conforman el sistema de visión. Dicha estructura almacena el nombre del objeto, un booleano que confirma si existe en la escena o no y la ruta para ejecutar el programa de reconocer objetos con los parámetros necesarios.

Esta estructura es usada en una función (**`rellenar_objetos()`**) que se ha creado para rellenar toda esa información dentro de un bucle de control *for*.

El programa, al iniciarse, la primera acción que realiza es la creación de un archivo de salida donde se guardarán las coordenadas de los objetos detectados (`centroides.txt`). Se crea la estructura de los objetos y se rellena con los valores por defecto.

A continuación, se realizan tantas llamadas iteradas al programa de reconocimiento de objetos como modelos tenga la base de datos. La finalidad de esta recursividad es que analice la escena comparando con cada uno de los modelos que se tienen y determine si el objeto del modelo existe en la escena. El siguiente paso a seguir es mostrarle al usuario los objetos que se han encontrado en la escena y preguntarle cuál desea recoger. Dependiendo de la respuesta obtenida se hace una llamada al sistema para ejecutar el programa de control del robot con unos parámetros u otros. Si de esta llamada se obtiene una respuesta satisfactoria se entrega un mensaje de que todo se ha llevado a cabo con éxito y si no, se muestra un mensaje de error.

### 6.3.2. Programa de obtención de imágenes.

Este programa genera una interfaz de visualización de la información que está captando el Kinect. Para ello crea una serie de objetos nube de puntos con información RGB en los cuales se guardará la escena que se desea capturar.

El visualizador incluye una serie de opciones para reiniciarlo, cerrarlo u obtener una imagen. Este último proceso se realiza cuando se pulsa la barra espaciadora. Cada vez que dicha tecla es pulsada el programa transforma la nube

de puntos asociada en archivo con extensión ".pcd" que guardará la información de la escena en la carpeta que se especifique en la ruta.

Este programa también puede utilizarse como visualizador de escenas que se hayan capturado con anterioridad, simplemente pasándole como parámetro "-v" y la ruta del archivo pcd. No obstante en el cd se encuentra un segundo programa desarrollado destinado a visualización puramente, **pcl_simple_viewer.**

Para más información de cómo funciona el código se recomienda ver el archivo cpp en detenimiento.

### 6.3.3. Programa de obtención de modelos individuales.

Para el diseño de este programa se han creado 3 funciones que son llamadas durante la ejecución del main.

La función **eliminar_elementos_fuera_mesa**() consigue quitar información irrelevante de la escena. Lo primero que hace es eliminar los puntos que quedan fuera de la nube de puntos en base a una distancia umbral que se le defina. A continuación le aplica un filtro a la imagen, el cual permite eliminar la información que no se encuentre en el rango de 0,9 a 1,3 m del sensor. Con esto se consigue acotar la escena a la ubicación espacial de la mesa. Por último, esta función guarda en fichero la nube de puntos filtrada.

La función **suavizar_nube**() recibe la nube de puntos y le aplica un suavizado en los puntos que la componen, eliminando el ruido que pueda presentar por errores de captación o limitaciones físicas del sensor. Así, en posteriores tratamientos se podrán extraer mejor los modelos individuales.

La función **count_Input_Files()** simplemente se encarga de saber cuántos archivos hay en un directorio que se le pase al programa como parámetro para saber cuántas veces debe ejecutarse el programa. Esto es muy útil pues permite trabajar con todos los archivos que queramos de una vez. Esta función únicamente funcionará bien si todos los archivos presentan el mismo nombre, por defecto “inputCloudX.pcd”, siendo X un número correlativo.

El programa comienza evaluando los archivos que tiene que tratar mediante la función anteriormente explicada y establece un bucle for con las repeticiones pertinentes. A continuación, se carga la nube de puntos y se le aplican los dos filtros que se han explicado al comienzo de este apartado.

Para poder separar objetos dentro de una escena es necesario hacer una segmentación. En este caso se ha llevado a cabo una segmentación RANSAC de tipo planar. Con ella se consigue eliminar el plano dominante de la escena, en este caso la mesa. En un principio, antes de diseñar la función **eliminar_elementos_fuera_mesa**() se tuvo que llevar a cabo más de una vez dicha segmentación para poder eliminar la pared y la mesa. Gracias a dicha función el código ha sido optimizado con respecto a su tiempo de ejecución.

Al final del programa se ha diseñado un algoritmo bastante complejo para el guardado de los clúster segmentados que los organiza en múltiples carpetas y les asigna nombre y numeraciones en concordancia a su ubicación, especificando el tipo de tratamiento que ha recibido la nube con objeto de poder analizar detalladamente el funcionamiento de cada uno de estos pasos.

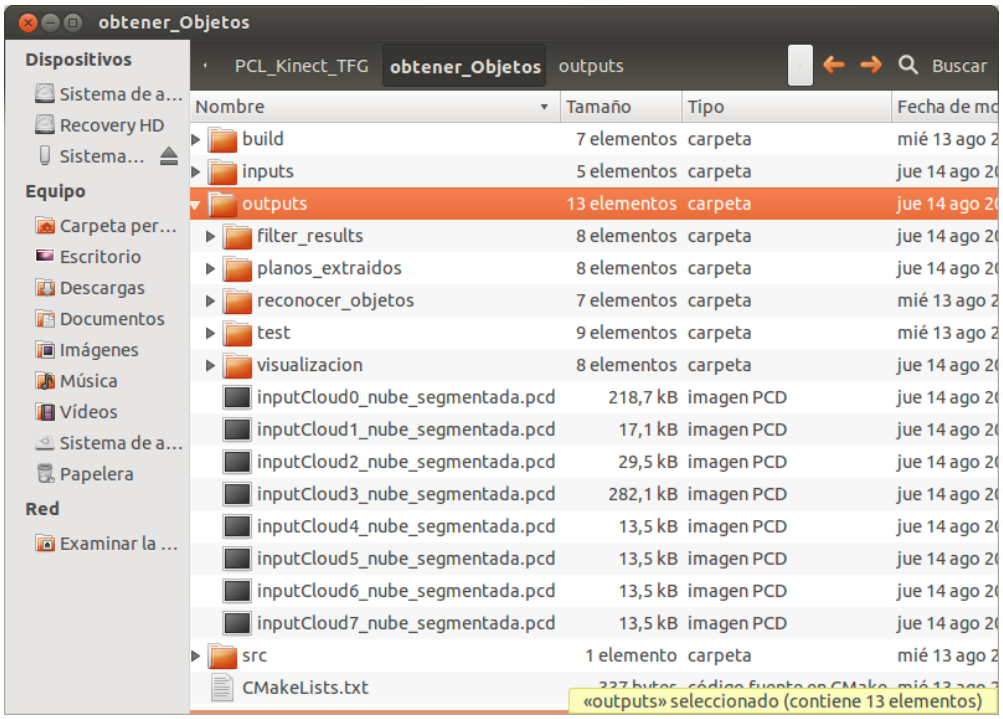



*Ilustración 35. Estructura de guardado de archivos.*

### 6.3.4. Programa de detección de objetos 3D.

Este programa se encarga de identificar los objetos presentes en una escena con los modelos obtenidos anteriormente y guardados en una base de datos.

El programa recibe como parámetros la escena que se vaya a analizar y el modelo que se quiere buscar. Para facilitar la identificación de la escena se le aplica un filtro en el cual se elimina el suelo y la pared del fondo, quedándose sólo la mesa con los objetos.

A continuación se calculan las normales de la superficie que generaría un punto con sus vecinos más próximos. También se disminuye la resolución, tanto a la escena como a los modelos, para agilizar los cálculos de la extracción de los puntos clave que es llevada a cabo justo después. Los puntos clave serán puntos

de referencia en la nube que permitirán compararla con otra para evaluar si existen coincidencias.

Los procesos de identificación de objetos necesitan de un descriptor que contenga las características del modelo que representan. Existen varios tipos de descriptores en función del algoritmo de identificación que utilizan. En este caso se utiliza el KdTree asociado a los puntos clave. Se buscan todos los vecinos cercanos que no exceden una distancia especificada y se guarda dicha información en un vector de correspondencias. Estas operaciones se llevan a cabo tanto en la escena como en el modelo facilitado.

Con este último proceso ya se puede proceder a la creación de los clústers. Los clústers pueden ser comparados por varios métodos. Durante la realización del proyecto se utilizó el algoritmo Hough 3D y el de consistencia geométrica, dando mejores resultados el segundo.

Por último, el programa muestra por pantalla los resultados obtenidos de la comparación de los clústers. Se indica si se han encontrado coincidencias con alguno de los objetos de la escena y en tal caso se calcula la matriz de rotación y traslación del objeto con respecto al modelo, es decir, muestra los giros y desplazamientos que ha habido que aplicarle al modelo para que se encaje en el objeto encontrado en la escena. Por defecto, los modelos se han generado con coordenadas de escena céntricas, esto es, se encuentran en el centro de la mesa alineados con el eje focal de la cámara. En una ventana se visualiza la escena con el objeto detectado en color rojo para que el usuario pueda comprobar que la detección se ha realizado correctamente.

La última operación que se lleva a cabo es la de calcular el centroide de la nube de puntos asociada al objeto detectado y guardar esa información en un fichero, que servirá de base de datos para el programa controlador de la mano.

El resultado que obtendríamos por pantalla tras la detección sería el siguiente:

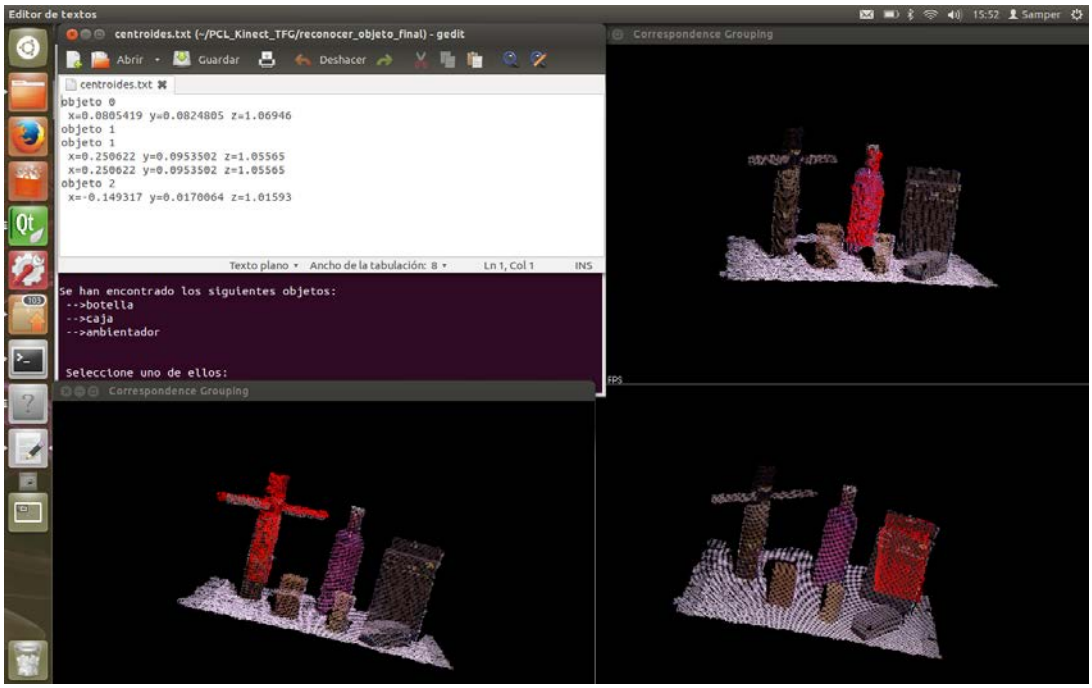



*Ilustración 36. Resultados de detección en 3D.*

A modo de conclusión del apartado, señalar que el índice de aciertos, una vez ajustados los parámetros es del 99 %. La única corrección o solución que tuvo que plantearse fue la incorporación de un segmento transversal al bote de ambientador ya que éste era confundido en numerosas ocasiones con la botella debido a la resolución y ruido del sensor.

## 6.4. Sincronización del robot manipulador Shadow Hand con el brazo robótico Robotnik LWA4P.

### 6.4.1. Presentación de librerías y principales funciones.

Una vez desarrollados y expuestos todos los módulos en los que se desglosa el presente Trabajo de Fin de Grado, queda por definir el nexo de unión que comunica todos estos paquetes: las librerías de comunicación y sincronización.

Para sincronizar ambos robots y comunicar los dos ordenadores responsables del control, se decidió desarrollar un conjunto de librerías que aglutinasen todas las funciones relativas al uso de sockets TCP/IP. Las pautas y premisas consideradas para el desarrollo fueron las siguientes:

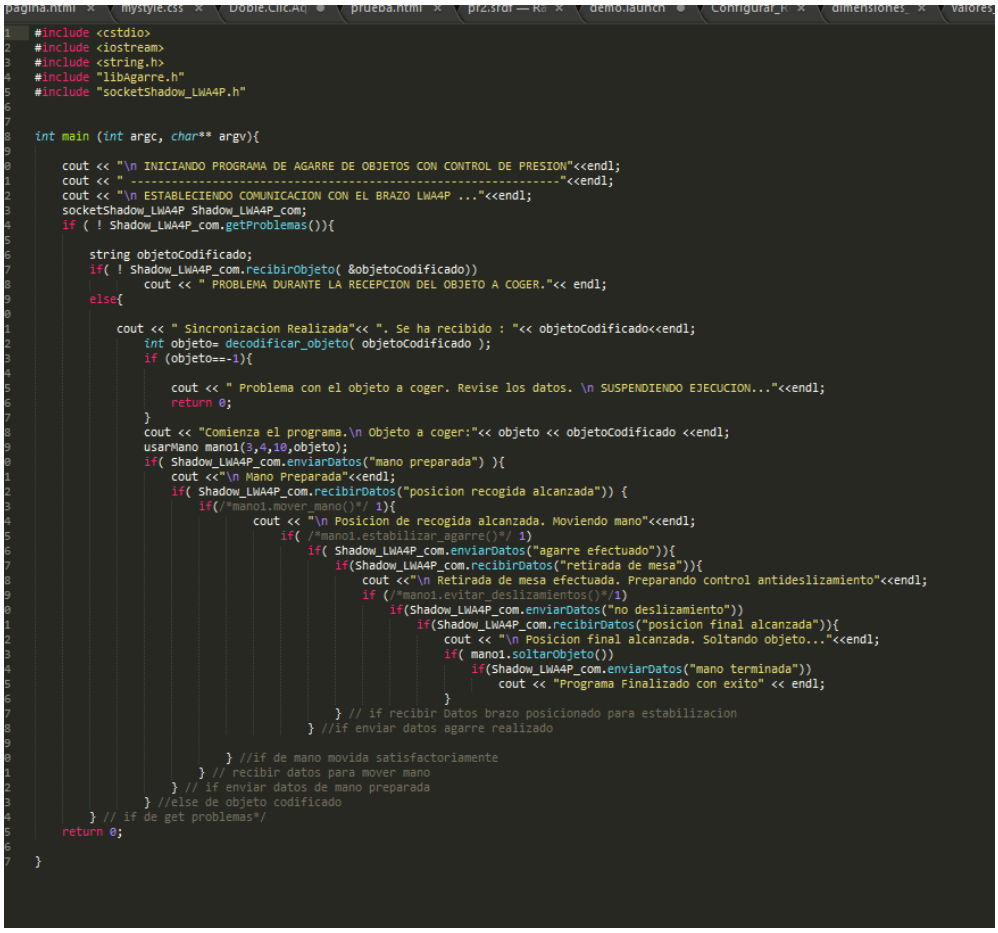
- Existen dos programas que involucran comunicaciones; uno de ellos en Debian, encargado de controlar la mano y otro, en Ubuntu, responsable del movimiento del brazo.
- El programa de Ubuntu realizará las operaciones que se consideren oportunas para adquirir los datos del sistema de visión artificial e informar al sistema de Debian del objeto que se debe agarrar.
- Deben definirse pautas y procedimientos que aseguren la sincronización y contemplen medidas de parada segura en caso de fallo.

Partiendo de estas hipótesis y principios de diseño, se generó una librería denominada **socketShadow_LWA4P**, con funciones para abrir sockets, establecer la comunicación con otro dispositivo y enviar/recibir datos. Aunque es posible encontrar todo el código en el cd adjunto, a continuación se comentarán las funciones y comportamientos más interesantes.

Llegados a este punto notar que se desarrolló un segundo programa denominado **Agarre_Presion_Sincro** encargado de adaptar el programa **Agarre_Presion** para trabajar con sincronizaciones.

Al iniciar el programa **Agarre_Presion_Sincro**, tras mostrar un mensaje de log se inicia el proceso de comunicación con el otro equipo. En caso de establecerse correctamente la comunicación, se da comienzo al proceso de sincronización y funcionamiento del programa global.

En la siguiente imagen se pueden observar los llamamientos que realiza el código fuente de este programa:



```
#include <cstdio>
#include <iostream>
#include <string.h>
#include "libAgarre.h"
#include "socketShadow_LWA4P.h"


int main (int argc, char** argv){

    cout << "\n INICIANDO PROGRAMA DE AGARRE DE OBJETOS CON CONTROL DE PRESION"<<endl;
    cout << " ------------------------------------------------------------"<<endl;
    cout << "\n ESTABLECIENDO COMUNICACION CON EL BRAZO LWA4P ..."<<endl;
    socketShadow_LWA4P Shadow_LWA4P_com;
    if ( ! Shadow_LWA4P_com.getProblemas()){

        string objetoCodificado;
        if( ! Shadow_LWA4P_com.recibirObjeto( &objetoCodificado))
                cout << " PROBLEMA DURANTE LA RECEPCION DEL OBJETO A COGER."<< endl;
        else{

            cout << " Sincronizacion Realizada"<< ". Se ha recibido : "<< objetoCodificado<<endl;
                int objeto= decodificar_objeto( objetoCodificado );
                if (objeto==-1){

                    cout << " Problema con el objeto a coger. Revise los datos. \n SUSPENDIENDO EJECUCION..."<<endl;
                    return 0;
                }
                cout << "Comienza el programa.\n Objeto a coger:"<< objeto << objetoCodificado <<endl;
                usarMano mano1(3,4,10,objeto);
                if( Shadow_LWA4P_com.enviarDatos("mano preparada") ){
                    cout <<"\n Mano Preparada"<<endl;
                    if( Shadow_LWA4P_com.recibirDatos("posicion recogida alcanzada")) {
                        if(/*mano1.mover_mano()*/ 1){
                                cout << "\n Posicion de recogida alcanzada. Moviendo mano"<<endl;
                                if( /*mano1.estabilizar_agarre()*/ 1)
                                    if( Shadow_LWA4P_com.enviarDatos("agarre efectuado")){
                                        if(Shadow_LWA4P_com.recibirDatos("retirada de mesa")){
                                            cout <<"\n Retirada de mesa efectuada. Preparando control antideslizamiento"<<endl;
                                            if (/*mano1.evitar_deslizamientos()*/1)
                                                if(Shadow_LWA4P_com.enviarDatos("no deslizamiento"))
                                                    if(Shadow_LWA4P_com.recibirDatos("posicion final alcanzada")){
                                                        cout << "\n Posicion final alcanzada. Soltando objeto..."<<endl;
                                                        if( mano1.soltarObjeto())
                                                            if(Shadow_LWA4P_com.enviarDatos("mano terminada"))
                                                                cout << "Programa Finalizado con exito" << endl;
                                                        }
                                        } // if recibir Datos brazo posicionado para estabilizacion
                                    } //if enviar datos agarre realizado

                            } //if de mano movida satisfactoriamente
                        } // recibir datos para mover mano
                    } // if enviar datos de mano preparada
                } //else de objeto codificado
        } // if de get problemas*/
    return 0;

}
```

*Ilustración 37. Código fuente del programa Agarre_Presion_Sincro.*

De observar el código se extrae que se emplean dos funciones principalmente:

- **enviarDatos()**. Encargada de enviar la cadena de datos recibida como argumento y esperar una confirmación. El valor de retorno es de tipo booleano e indica si la confirmación ha sido recibida o no.
- **recibirDatos()**. Encargada de gestionar la recepción de datos. Esta función espera recibir una cadena de datos coincidente con el string recibido como argumento. El valor de retorno es de tipo booleano, devolviéndose false si la cadena recibida no coincide con la esperada o si concluye el tiempo máximo de espera para la recepción.

### 6.4.2. Flujo de Funcionamiento.

Se tiene que el flujo de trabajo y funcionamiento es el siguiente:

1. Establecimiento de la comunicación e inicio de las sincronizaciones (todas las funciones necesarias las realiza el constructor de la clase, siendo necesario únicamente verificar el valor de la variable booleana problemas_comunicación a través de la función **getProblemas()**).
2. El Software de control del brazo recibe la información, del sistema de visión, sobre el objeto a coger (dimensiones, posición en el espacio…) y envía el tipo de objeto a través del socket abierto (función **enviarDatos()**).
3. El Software de control de la mano recibe los datos sobre el objeto a agarrar y verifica si dicho objeto se encuentra en la base de datos de objetos "agarrables o conocidos" (fichero dimensiones_objeto.txt). Este proceso de recepción extraordinaria y verificación se realiza por medio de las funciones **recibirObjeto**() y **decodificar_objeto**().
4. El robot manipulador Shadow Hand inicia los procesos de lectura de sensores y verificación de actuadores.
5. Una vez preparada, la mano comunica al programa de control del brazo su estado: "*mano preparada*".
6. Tras recibir la orden anterior, el brazo posiciona al manipulador para que realice el agarre, enviando "*posición recogida alcanzada*" cuando ha finalizado el posicionamiento del efector final.
7. Conocida su correcta posición, la Shadow Hand inicia el proceso de agarre, siguiendo los valores del algoritmo cinemático y estabilizando la presión ejercida acorde a unos valores de referencia.
8. El software de control del manipulador envía al controlador del brazo la sentencia: *"agarre efectuado"*. (**función enviarDatos ()**).
9. El brazo LWA4P eleva la mano dos centímetros y la retrae 20 cm para que el objeto quede en suspensión.
10. Concluido el movimiento 9, el software de control del brazo comunica su estado: *"retirada de mesa"*.
11. El programa encargado de controlar la mano inicia los controles antideslizamiento.

12. Una vez estabilizado y asegurado el agarre, el brazo es informado de la situación: "*no deslizamiento*"
13. A continuación, el brazo se desplaza a la posición de entrega del objeto y señala al software de control de la mano que debe soltar el objeto: "*posición final alcanzada*".
14. La mano procede a soltar el objeto y adoptar la posición inicial. Tras finalizar estas tareas, este módulo del programa global envía "mano terminada" antes detenerse.
15. Una vez recibido el mensaje anterior, el robot adopta la posición inicial, quedando preparado para repetir el proceso.
16. Detención de los motores y fin del programa.

En el siguiente esquema se resumen todos estos pasos, contemplándose únicamente los fallos debidos a comunicaciones.

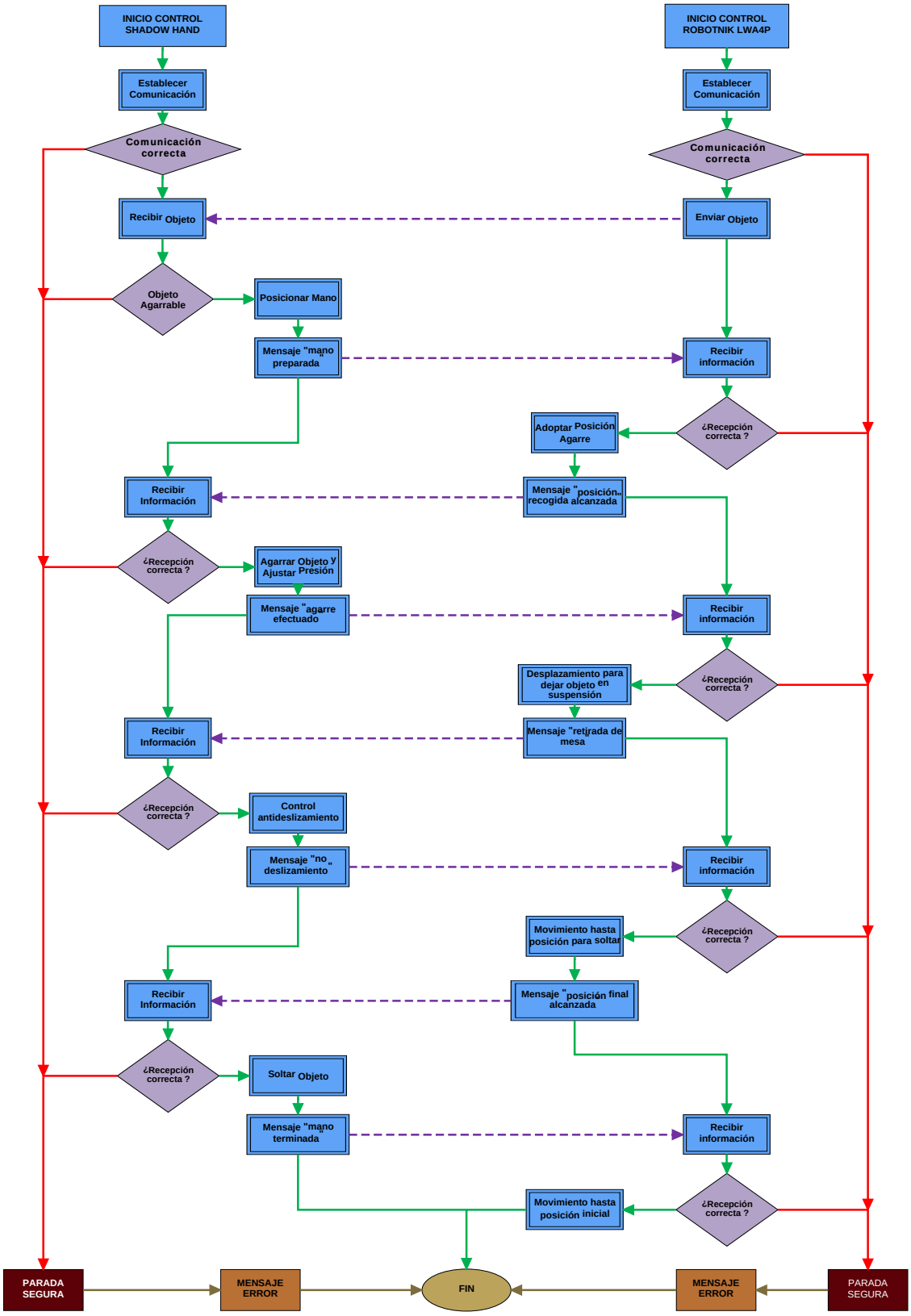



*Ilustración 38. Esquema de funcionamiento de las sincronizaciones.*

### 6.4.3. Medidas de seguridad y comprobaciones.

Como se puede observar en la imagen del código, todas las funciones devuelven un valor booleano. En caso de que algún procedimiento o paso de los citados en el apartado anterior falle, se dará inicio a un protocolo de finalización consistente en mostrar un mensaje de error y detener el módulo problemático (controlador de la mano o controlador del brazo).

Las comunicaciones se han establecido de modo tal que todos los envíos y recepciones verifiquen el estado del socket. De este modo, si alguno de los extremos se detuviese, el extremo contrario sería consciente de que ha acontecido algún problema e iniciaría una parada segura. Para ello:

- En las funciones de envío de datos se espera que el receptor envíe una confirmación (valor true) en menos de 1 segundo para informar de que no ha habido problemas. En caso de no recibirse, la función devolvería false, provocando que el programa usuario inicie tareas de parada.
- En las funciones de recepción de datos, puesto que se conocen los mensajes a enviar, se comprueba la validez de la información recibida por medio de comparaciones. En caso de recibirse información incompleta o datos inesperados, la función finaliza indicando la existencia de problemas.

# CAPÍTULO 7.
# HARDWARE DESAROLLADO

Para unir la mano al brazo robótico no existía ningún soporte ni anclaje, ya que estos son de fabricantes distintos. Por ello, fue necesario que los desarrolladores encargados de controlar los robots debatiesen las posibilidades y opciones existentes para realizar el acople que actuaría como nexo de unión entre ambos robots.

A partir de la información técnica de que se disponía, planos de la base de la Shadow Hand, una vista en planta de la situación de los tornillos y brida para el anclaje de herramientas en el extremo del brazo, se decidió elaborar un anclaje que consistiése en una extensión de la brida añadiendo en uno de sus extremos una forma de sujeción de la mano.

*Ilustración 39. Brida del brazo LWA4P.*

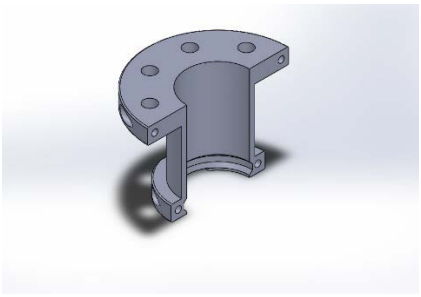


*Ilustración 40.Diseño en 3D del acople desarrollado.*

A pesar del acabado conseguido y procesos de pulido a los que ésta fue sometida posteriormente, presenta un problema. El peso de la mano es excesivo para ser anclado al diseño de Schunk, lo que provoca que, la parte superior del acople, se incruste en exceso en la articulación 6 del brazo, bloqueando el movimiento de la misma. Este problema trató de superarse sometiendo a la pieza a procesos de refrentado y pulido que permitiesen un acople más suave. Sin embargo, sólo se consiguió retrasar el fenómeno. De este modo, se ha optado por acoplar la mano únicamente cuando es necesario usar el conjunto.

El segundo acoplé desarrollado consistía en un soporte de altura e inclinación variable para la cámara. El objetivo era anclar la cámara a la base del robot y estudiar varias opciones que permitiesen capturar mejores vistas del objeto, de modo que nó solo se reconociese el objeto sino que, además, se extrajeran todas sus dimensiones. Aunque el soporte fue diseñado y encargado, no se recibió con tiempo suficiente.

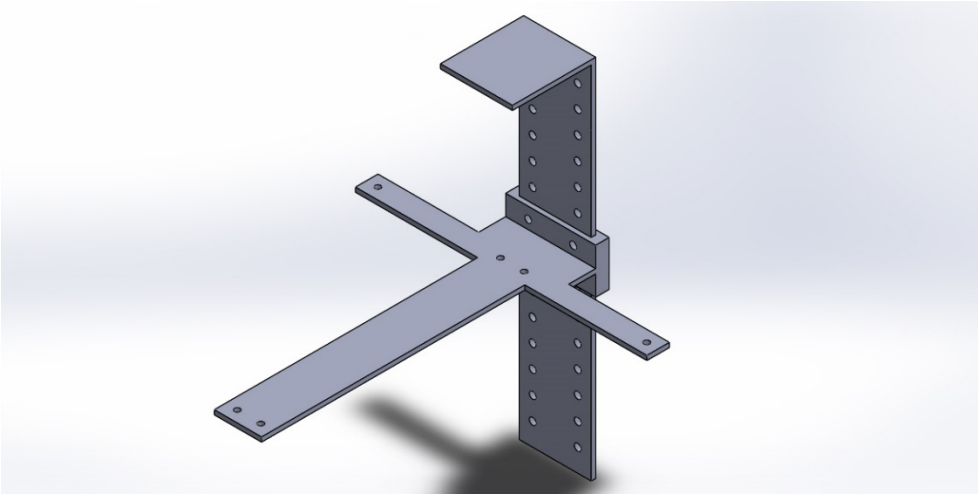


*Ilustración 41. Modelo 3D del soporte para la cámara con capacidad para añadir un cabezal pivotante.*

Todos los planos pueden ser revisados en el *Anexo 12. Planos de Acoples diseñados,* destacar que fueron diseñados por el alumno y compañero de laboratorio Joaquín Macanás Valera el cual hizo gala de sus conocimientos de diseño en SolidWorks.

# CAPÍTULO 8.
# FUTUROS DESARROLLOS.

Para dar por terminada la presente memoria del Trabajo de Fin de Grado se proporcionarán pautas e ideas con las que continuar las investigaciones y desarrollos realizados con el proyecto que nos atañe. Antes de enumerar nuevos recorridos y plantear la realización de desarrollos más complejos que asienten su base en los conceptos descritos a lo largo de estas páginas, se considera importante mejorar algunos aspectos y características:

- Mejora del sistema de visión artificial para reconocer y dimensionar totalmente los objetos detectados.
- Crear una interfaz gráfica con la que comunicarse con los robots e integrar en ésta todos los mensajes de error y avisos. Se recomienda seguir los principios del desarrollo de Scadas.
- Desarrollo de algoritmos de evitación de obstáculos en las trayectorias.
- Desarrollar un protocolo de comunicación más sólido, embebido en ROS.
- Intentar proporcionar al software de control de la mano comportamientos que permitan el autoaprendizaje para lograr agarres estables.

Por otro lado, surgen numerosas ideas que parten de las bases establecidas por este proyecto, de entre las que destacan:

- Uso de un sensor LeapMotion que permita que el robot copie los movimientos realizados por un usuario. Este desarrollo supone un trabajo gemelo al presentado ya que sustituye los sistemas de visión artificial y algoritmos de movimiento por acciones de captación de movimientos humanos y emulación de los mismos.
- Trabajo cooperativo entres dos brazos manipuladores (brazo más herramienta de manipulación).
- Adaptación del trabajo para agarrar objetos dinámicos o en movimiento.
- Adhesión de un escáner a un brazo robótico para capturar el entorno y generar mapas tridimensionales. En colaboración podría trabajar un segundo brazo con un manipulador que agarrase objetos escondidos o de difícil acceso.
- Crear IronMan.

# ANEXOS

# Anexo 1. Instalación de Ubuntu.

## 1.1. Escogiendo la versión.

Una de la principales diferencias entre el código libre y el comercial es la innumerable cantidad de versiones y actualizaciones que saca el primero de ellos (concretamente Ubuntu, suele cambiar de versión cada 6 meses). Antes de debatir qué versión debemos emplear, es necesario realizar una distinción. Ubuntu se caracteriza por tener dos tipos de versiones, la habitual, lanzada cada 6 meses y cuyo soporte aspira al año de vida (o dos en algunos casos excepcionales) cuya nomenclatura responde a la forma: *Ubuntu – tipo (desktop, server…) – XX.X.X*. A todo buen observador no se le habrán escapado una serie de datos curiosos de esta nomenclatura; entre otras, tras la palabra Ubuntu, se hace referencia a "tipo". Este hecho puede llevar a plantear la siguiente pregunta conductora: ¿Existen, por lo tanto, distintas versiones y tipos de Ubuntu? No del todo, cada cierto tiempo aparece una versión nueva de Ubuntu, ésta se encuentra disponible en distintos formatos o formas, distinguiéndose principalmente entre: completa (desktop version) y recortes o adaptaciones de la misma para usar en servidores (server versión, sin interfaz gráfica), para móviles (mobile versión)… La combinación de letras "X" que sigue al tipo de versión, hace referencia a: Versión general, modificación de esa versión y sub-modificación. Lo que expresado en otros términos, puede definirse señalando que cada cambio importante viene marcado por una versión general, mientras que las depuraciones y correcciones realizadas, pero que no conllevan cambios de interés en el entorno, se muestran como partes de dicha versión general y por último, las pequeñas correcciones de bugs y errores sin importancia se añaden como sub-modificaciones de la antes citada. Así, por ejemplo:

- Ubuntu-desktop 13.1.0. Primera modificación o corrección de la versión general 13 completa.

- Ubuntu-server 13.1.2. Versión anterior pero, en este caso, para ser usada en servidores y con dos pequeñas correcciones de bugs realizadas.
- Ubuntu 14.0.1. Referencia a la nueva versión desarrollada, con cambios interesantes respectos a las versiones 13.X.

Otro aspecto a tener en cuenta al instalar una versión es el tiempo de mantenimiento y estabilidad de las mismas. Cada 2 años aparece una versión LTS (Long Time Supported), cuyo mantenimiento está asegurado durante 5 años. Además, estas versiones destacan por su estabilidad respecto a las "normales", lanzadas cada 6 meses y con escaso mantenimiento, que destacan por ser empleadas como fuentes de pruebas y debates acerca de las nuevas funcionalidades a incorporar en la próxima versión LTS. De este modo, es aconsejable instalar una versión LTS, aunque suponga poseer menores mejoras gráficas u opciones de configuración.

Una vez aclarado el tema de las distintas versiones y sub-clasificaciones de las mismas que es posible instalar, ha llegado el momento de analizar cuál de ellas seleccionar. Aunque los más innovadores y atrevidos sugerirían abalanzarse, sin duda alguna, sobre la última versión lanzada; mientras que, por su parte, los conservadores apuntarían hacia la última LTS distribuida, ambos olvidan una parte importante y fundamental de este trabajo: ROS se ejecuta sobre Ubuntu y utiliza gran parte de su Kernel. Es en este momento en el que debemos decidir, en primer lugar que versión de ROS necesitamos (como se puede ver, a los desarrolladores de software libre les encantan las versiones, aunque como se verá posteriormente, las de ROS son las más carismáticas y entrañables). Por norma general, para tomar esta decisión, el lector novel debe navegar por los foros y páginas relacionadas con ROS durante unas horas en busca de la información pertinente que le permita realizar un correcto análisis crítico de las opciones que a su trabajo pueden ofrecer las distintas versiones. Sin embargo, en este anexo se tratará de facilitar esta ardua búsqueda: en este caso, se decidió instalar *hydro* ya que *Jade* estaba por desarrollar, *Indigo* aún mostraba evidentes necesidades de maduración (faltaban numerosos desarrollos por implementar) y *groovy* no terminaba de hacer mella en Shadow (siendo sus paquetes imprescindibles para llegar a buen fin). No obstante, durante la instalación de ROS y sus dependencias

se abordarán las instalaciones en *Hydro* y *Groovy*, ya que; en ciertos momentos puntuales, se realizaron pruebas y comparaciones entre versiones, dado que el control para el brazo robot LWA4P sólo se encontraba disponible en Groovy.

Una vez escogida la versión de ROS a instalar, nos dirigimos a la página oficial en la que se ofrece una guía sobre todo el procedimiento de instalación, para *hydro*: http://wiki.ros.org/hydro/Installation/Ubuntu. En caso de escoger una versión diferente, basta con cambiar "/hydro" por el nombre de la versión en cuestión ( /groovy, /indigo, /jade …). En esta página aparecen una serie de comandos que, una vez instalado Ubuntu, se siguen, como si de un acto de fé se tratase, y resultan en la instalación del sistema. Entre esta serie de comandos se encuentra un conjunto de ellos destinado a configurar los repositorios de Linux según la versión de Ubuntu que se tenga instalada instalada:

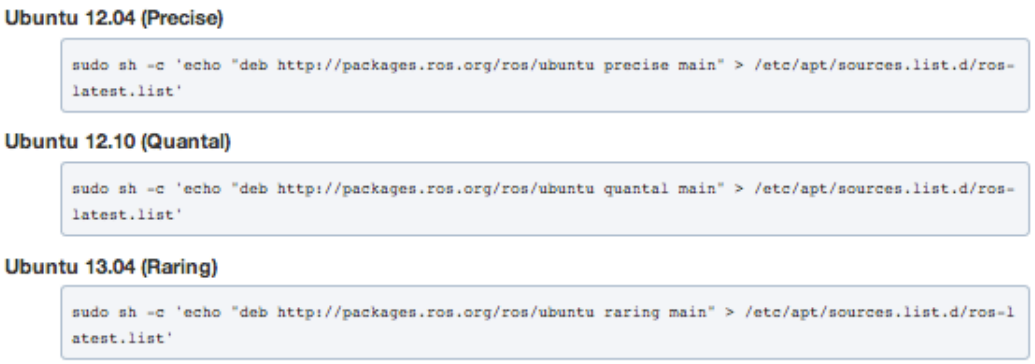



*Ilustración 42. Versiones de Ubuntu en las que instalar ROS Hydro*

Esto supone, por tanto, una gran limitación. Así, si se desea instalar ROS Hydro, es necesario contar con Ubuntu 12.04, 12.10 o 13.04. Una vez conocidas las versiones de Ubuntu compatibles, nos dirigimos a la página oficial del sistema operativo: http://www.ubuntu.com/download, al pinchar sobre el apartado versión desktop, nos direccionarán a la siguiente página:

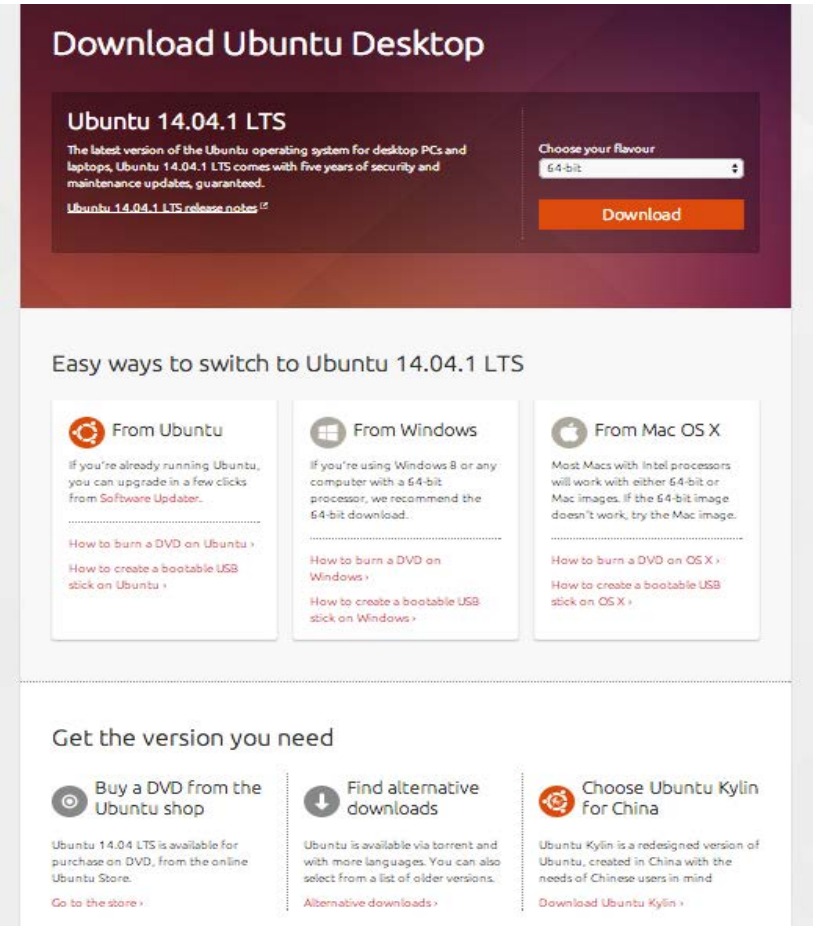
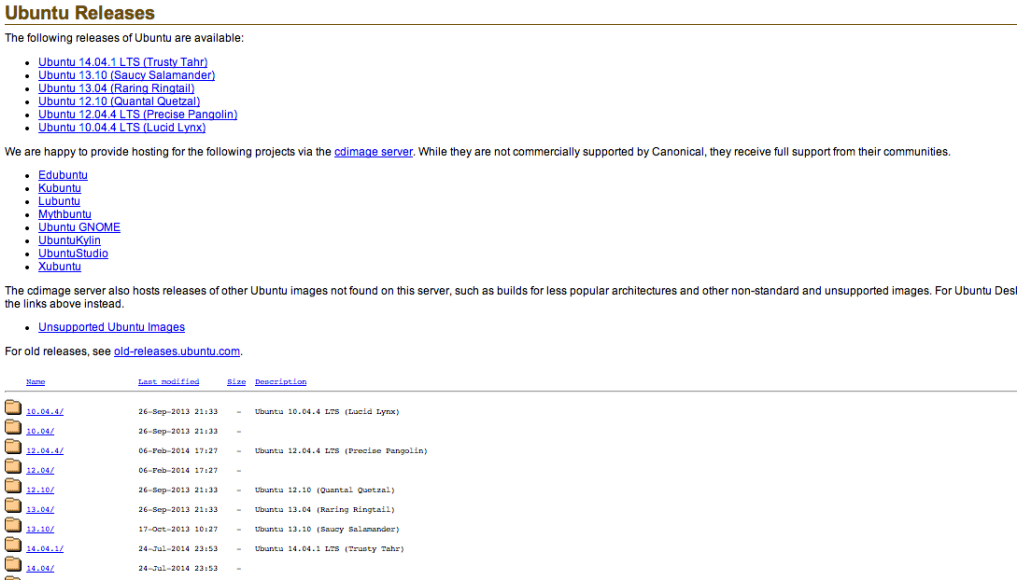



*Ilustración 18. Página versiones de Ubuntu.*

Puesto que lo usual, es que no se desee instalar la última versión lanzada (carencia de versiones de ROS útiles compatibles), en la parte inferior clicamos en el apartado "Alternative Downloads". Aparecerá así una lista de todas las versiones de Ubuntu (http://releases.ubuntu.com/)



**Ubuntu Releases**

The following releases of Ubuntu are available:

- Ubuntu 14.04.1 LTS (Trusty Tahr)
- Ubuntu 13.10 (Saucy Salamander)
- Ubuntu 13.04 (Raring Ringtail)
- Ubuntu 12.10 (Quantal Quetzal)
- Ubuntu 12.04.4 LTS (Precise Pangolin)
- Ubuntu 10.04.4 LTS (Lucid Lynx)

We are happy to provide hosting for the following projects via the cdimage server. While they are not commercially supported by Canonical, they receive full support from their communities.

- Edubuntu
- Kubuntu
- Lubuntu
- Mythbuntu
- Ubuntu GNOME
- UbuntuKylin
- UbuntuStudio
- Xubuntu

The cdimage server also hosts releases of other Ubuntu images not found on this server, such as builds for less popular architectures and other non-standard and unsupported images. For Ubuntu Desl the links above instead.

- Unsupported Ubuntu Images

For old releases, see old-releases.ubuntu.com.

| Name | Last modified | Size | Description |
|---|---|---|---|
| 10.04.4/ | 26-Sep-2013 21:33 | - | Ubuntu 10.04.4 LTS (Lucid Lynx) |
| 10.04/ | 26-Sep-2013 21:33 | - | |
| 12.04.4/ | 06-Feb-2014 17:27 | - | Ubuntu 12.04.4 LTS (Precise Pangolin) |
| 12.04/ | 06-Feb-2014 17:27 | - | |
| 12.10/ | 26-Sep-2013 21:33 | - | Ubuntu 12.10 (Quantal Quetzal) |
| 13.04/ | 26-Sep-2013 21:33 | - | Ubuntu 13.04 (Raring Ringtail) |
| 13.10/ | 17-Oct-2013 10:27 | - | Ubuntu 13.10 (Saucy Salamander) |
| 14.04.1/ | 24-Jul-2014 23:53 | - | Ubuntu 14.04.1 LTS (Trusty Tahr) |
| 14.04/ | 24-Jul-2014 23:53 | - | |

*Ilustración 18. Página distintas Releases disponibles.*

Podemos comprobar que de las versiones necesarias para ROS *Hydro*, la única LTS es la 12.04, siendo ésta la que conviene instalar. Una vez descargada la imagen, grabada en disco o USB-bootable y configurada la BIOS para arrancar desde el medio adecuado se plantean dos opciones: Formatear todo el disco duro del ordenador o, como en la mayoría de los casos, realizar partición. Ambos casos se resumen en seguir los pasos específicos del lanzador con la salvedad de clicar en formatear todo el disco o instalación conjunta con Windows / Mac. Sin embargo, si se busca instalar Ubuntu en un "ordenador de la manzana", deben de tomarse una serie de medidas previamente.

## 1.2. Instalación Híbrida (Particionado) en Mac.

Antes de comenzar esta explicación, es imperativo señalar que es necesario llevar mucho cuidado durante la realización de este proceso, siendo conveniente salvar la información de interés de nuestro sistema en una copia de seguridad (la aplicación TimeMachine permite gestionar copias de seguridad de forma cómoda e intuitiva). El problema principal no se localiza en el proceso de instalación del sistema sino en la gestión del mismo. Por ello, se anima, a todo usuario novel, a probarlo y familiarizarse con él antes de proceder a su instalación física.

También se debe señalar que todos los pasos aquí mencionados tratan de servir de referencia o pequeña guía, algunos de ellos simplemente se mencionarán dada su simpleza de ejecución, pudiendo encontrarse gran cantidad de tutoriales en la web sobre todos los aspectos del proceso. De igual modo, tratar de reproducirlos queda a responsabilidad del lector, el que bajo ningún concepto debe señalar como responsable del deterioro o posibles fallos en el equipo al autor del presente TFG.

Otro aspecto importante a tener en cuenta se encuentra relacionado con la compatibilidad hardware, el hardware de Apple es muy cerrado y ciertas versiones de Ubuntu, según se comprobó a lo largo de la realización del presente trabajo, no terminan de adaptarse a él. Este hecho produce, en el mejor de los casos, la pérdida de alguna funcionalidad del teclado. En la siguiente página de la comunidad de Ubuntu (https://help.ubuntu.com/community/MacBookPro) se indica qué versiones han sido probadas satisfactoriamente en distintos ordenadores de la "marca de la manzana". Para la realización de este TFG, ninguna versión fue satisfactoria por lo que se probaron varias versiones, lo que permitió conocer el proceso de instalación de Ubuntu con bastante profundidad. Los resultados obtenidos para el modelo empleado, MacBook Pro de 15" de principios de 2011 (modelo MacBook Pro 8,2) fueron:

- Ubuntu 12.04 LTS. Resultados satisfactorios aunque algunas funcionalidades del trackpad como pulsación suave no quedan implementadas directamente y deben modificarse manualmente.

(Fue la que mejor resultados proporcionó por lo que se usó como definitiva).

- Ubuntu 13.04. Funciona correctamente aunque presenta cierta inestabilidad al arrancar Gazebo.
- Ubuntu 13.10. La pantalla no llegó a ser detectada.
- Ubuntu 14.04 LTS. Los drivers de la gráfica no estaban implementados cuando fue probada.

Todas estas pruebas fueron realizadas en máquina virtual, dada la comodidad que ofrecen para desechar sistemas que no funcionan evitando daños colaterales.

Si eres usuario de Mac estarás acostumbrado a tener que realizar pasos "especiales" cuando quieres instalar algo. Por supuesto, este caso no iba a ser menos. Los gestores de arranque y Bios de Mac son ligeramente distintos a los del resto de sistemas por ciertas características que, en su momento, quiso destacar Apple. Entre ellas se encuentra su rapidez en el arranque. De este modo y a pesar de implementar Bootcamp entre su software básico, la instalación de Linux requiere de pasos especiales. No debemos olvidar que estamos instalando dos sistemas basados en Unix.

El primer cambio consiste en realizar una partición con el espacio que deseemos entregar a Ubuntu para ello se puede utilizar la utilidad "Asistente de BootCamp" que viene con el CD de aplicaciones entregado junto al ordenador, o bien con "Utilidad de discos", la cual es posible encontrar en la carpeta Utilidades. Con BootCamp el proceso es bastante más sencillo ya que queda reducido a desplazar una barra.

El siguiente paso es modificar la Bios y su gestor de arranque, ésto que parece algo descabellado cuando se lee por primera vez se traduce en instalar un software, que mostrará, al arrancar el ordenador, las distintas opciones de arranque disponibles en el sistema, ya sea desde partición, CD o USB "booteable"… Para ello nos dirigimos a la página de ReFiT (http://refit.sourceforge.net/#download) y descargamos la versión para Mac OS X, cuando tengamos el archivo .dmg bastará con montarlo y hacer doble click en el paquete *rEFit.pkg*.

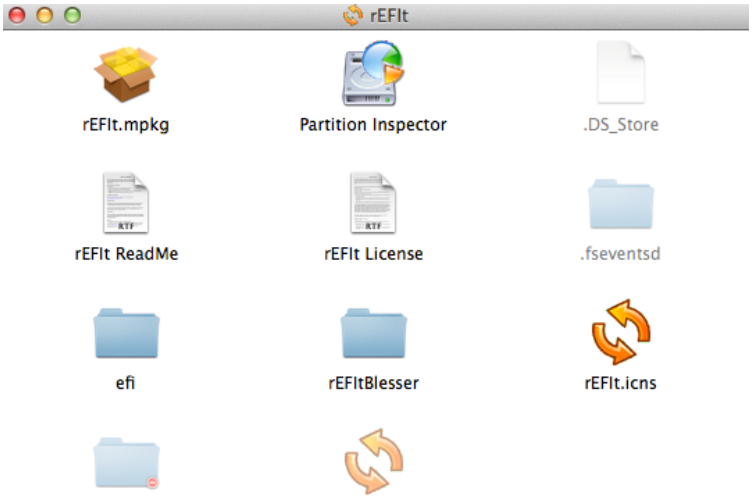



*Ilustración 19. Contenido del .dgm de rEFIt.*

A continuación, se abrirá un instalador que nos guiará durante todo el proceso de instalación. Una vez finalizado y creado el dispositivo desde el que arrancaremos Ubuntu para instalarlo, debemos comprobar que la instalación ha sido correcta y podemos proseguir con el proceso, para ello es necesario reiniciar y verificar que al arrancar aparece un nuevo menú en el que nos dan a elegir entre distintas opciones de arranque.


*Ilustración 44. Ejemplo del menú de arranque rEFit.*

Por último, seleccionamos arrancar desde el dispositivo con la imagen de Ubuntu, cuyo icono corresponde al del Pingüino (desde USB pueden aparecer tres opciones de lanzado, sólo una de ellas funcionará mientras que las otras conducirán a una pantalla negra o estado "standby" del que deberá recuperarse "manualmente"). A diferencia de las otras instalaciones en las que se escoge todo automático y nos dejamos guiar por el instalador, en Mac debemos molestarnos algo más. Al iniciar el dispositivo de imagen se nos dará a elegir entre varias opciones, una de ellas será "Probar Ubuntu", seleccionamos ésta. Una vez en ella, veremos Ubuntu como quedará en el equipo (aunque con cierta carencia de controladores específicos y retardo), hacemos click en el icono superior de la barra lateral y buscamos al aplicación GParted (Asistente de Particiones de Ubuntu). Nos aparecerán todas las particiones del equipo, entre las que figura una sin formato (o con el formato que hayamos dado al particionar en Mac, NTFS si se realizó con BootCamp). Esta última partición debe dividirse en una de 1Gb de tipo Linux-swap y otra (con todo el espacio restante) de formato Ext4. Para concluir, nos fijamos en el nombre de esta última que será del tipo div/sdaX (siendo X un número) y le damos al único icono del escritorio: Instalar Ubuntu. Seguimos el procedimiento habitual, seleccionando instalación compartida con Mac y nos fijamos que la partición en la que se instala Ubuntu se corresponde con la de extensión Ext4 que acabamos de crear. Si todo va bien, al reiniciar tendremos un sistema híbrido Unix.

Para concluir, como curiosidad señalar que también existe la posibilidad de crear un arranque triple (Mac, Windows y Linux). Sin embargo, es necesario mencionar que éste hace uso de una funcionalidad integrada por Apple que permite crear más particiones de las que soporta físicamente la Bios sin caer en la ralentización que introducen las particiones secundarias realizadas tradicionalmente. Se deja a manos del lector, la tarea de indagar más sobre este método y sus posibilidades.

# Anexo 2. Primeros Pasos en Ubuntu.

Por experiencia, se aconseja tratar de familiarizarse con este nuevo sistema operativo, ya que supone un cambio drástico respecto a los sistemas habitualmente usados: Mac y Windows. Aunque permite mayor libertad y ofrece evidentes mejoras en cuanto a rendimiento y potencial, la curva de aprendizaje, para el disgusto de todos, es un tanto escarpada. Para aquellos que vean Ubuntu como un comienzo muy abrupto, a pesar de ser una de las distribuciones más similares a los sistemas amigables ya citados, se recomienda iniciarse con una distribución relativamente reciente conocida como Linux Mint que trata de facilitar la transición a Linux.

A continuación, se describirán un conjunto de aspectos del sistema y dará una introducción al uso de comandos por consola, herramienta fundamental de todo usuario de Linux. Por último, para concluir este anexo, se darán referencias para la creación de MakeFiles y una lista de comandos útiles.

## 2.1. Presentación del entorno.

El entorno, a primera vista, es muy similar a los del resto de sistemas, constando de una pantalla de inicio de sesión desarrollada en Unity, con objeto de ser ligera y rápida, y un escritorio con barras de herramientas, información de estado y acceso a las aplicaciones.

Sin embargo, antes de comenzar a trastear, resulta conveniente terminar de configurar el sistema. En primer lugar, pinchamos sobre el botón de configuración que aparece en la "*barra de aplicaciones*", a la izquierda. A continuación, se desplegará una ventana con múltiples opciones de configuración. De todas ellas, debemos prestar atención a hardware específico (primer icono verde del apartado hardware) para instalar los drivers privativos necesarios. El

siguiente paso será configurar el idioma, en caso de que no se haya hecho ya, para ello pinchamos en soporte del lenguaje y escogemos el idioma deseado. Y finalmente, en ajustes del ratón, dónde podremos configurar la velocidad y sensibilidad del trackpad y ratón. En los macbook se puede conseguir una experiencia con el trackpad muy similar a la del sistema de Apple, cambiando la sensibilidad y habilitando los scrolls.


*Ilustración 45. Ventana de configuración de Ubuntu.*

Para concluir la configuración, se recomienda abrir el terminal y, disponiendo de conexión a internet, teclear dos comandos:

Sudo apt-get update

Sudo apt-get upgrade

Con el primero se recopila la información de todos los aspectos que tenemos en el sistema y comprueba si existen versiones nuevas. Con el segundo, se realizan todas las actualizaciones necesarias. El comando sudo nos permite realizar tareas con permisos de superusuario, como pueden ser modificar los ficheros del kernel de Linux o de configuración del sistema.

Este último paso es recomendable realizarlo cada vez que arranquemos Linux con objeto de mantener el sistema lo más actualizado posible. El resto de acciones típicas de interactuación gráfica como crear carpetas, guardar documentos, copiar ficheros… se realizan de forma similar a Windows. Por ello, a partir de aquí, se deja a cuenta del usuario comenzar a familiarizarse con el entorno.

En cuanto a la instalación de aplicaciones, ésta se puede realizar de diversas maneras: por consola o terminal (la más usual) y/o mediante el gestor de software, cuyo icono se puede encontrar en la "*barra de aplicaciones*" que aparece por defecto a la izquierda.

## 2.2. El terminal de Ubuntu: presentación y comandos.

El terminal es la herramienta fundamental y principal para interactuar con el sistema operativo en cualquier sistema Linux. Aunque, en un principio esto puede parecer engorroso, una vez acostumbrados a él veremos que la comodidad y velocidad que ofrece para instalar nuevas aplicaciones y gestionar el sistema es muy elevada. Llegados a este punto, señalar que en la tienda de aplicaciones se pueden encontrar gran variedad de aplicaciones de terminal que permiten modificar su aspecto (color de fondo, tamaño de la letra, tipo de fuente…).

En primer lugar, conviene citar las distintas formas en las que se puede acceder al terminal:

- En cualquier GNU/Linux tenemos la llamada terminal o consola que abre un shell o intérprete de comandos. En Ubuntu se abre buscando en el Dash o tablero de Unity: "**Terminal**" o pulsando la combinación de teclas `Ctrl`+`Alt`+`T`

- También se puede pasar al **modo texto** (intérprete de comandos) desde el modo gráfico pulsando las teclas: Ctrl+`Alt`+`F1` o bien con: `F2 F3 F4 F5 F6`.
  Esto hace que el sistema salga del modo gráfico y acceda a alguna de las seis consolas virtuales de Linux, a las cuales también se puede acceder cuando se arranca en modo de texto.
  Para volver al modo gráfico hay que presionar `Ctrl`+`Alt`+`F7` o `Ctrl`+`Alt`+`F8` (Según la sesión en modo gráfico a la que deseemos regresar).

En segundo lugar, es necesario conocer una serie de normas básicas referidas a la nomenclatura empleada al trabajar en modo texto:

- Las aplicaciones con nombres compuestos se escriben con guion entre las palabras (ej. compizconfig-settings-manager).

- Para los nombres de archivos y directorios que contienen espacios en blanco hay que envolverlos en comillas dobles (ej. "nombre archivo") o simples (ej. 'nombre archivo').

- Los espacios en blanco se utilizan únicamente para separar órdenes (ej. para instalar varios paquetes: `sudo apt-get install avidemux k3b kde-i18n-es k3b-i18n`, vemos que dichos paquetes están separados por espacios en blanco entre ellos).

- La ruta "/home/tu_usuario" se puede cambiar por el símbolo "~", que viene a sustituirlo en la línea de órdenes, sea cual sea el nombre del usuario.

- Los comandos hay que teclearlos exactamente.

- Nomenclatura Case-Sensitive. Las letras mayúsculas y minúsculas se consideran como diferentes.

- En su forma más habitual, el sistema operativo utiliza un signo de $ como *prompt* para indicar que está preparado para aceptar comandos, aunque este carácter puede ser fácilmente sustituido por otro u otros elegidos por el usuario. En el caso de que el usuario acceda como administrador este signo se sustituye por #.

- Cuando sea necesario introducir el nombre de un fichero o directorio como argumento a un comando, Linux, permite escribir las primeras letras del mismo y realiza un autorrellenado al presionar la tecla del tabulador. Si no puede distinguir entre diversos casos rellenará hasta el punto en el que se diferencien.

## 2.3. Elaboración de MakeFiles. Compilación en Linux.

En contraste con las opciones habituales usadas en sistemas operativos no basados en Linux para la compilación, en las distintas distribuciones derivadas de este núcleo predomina el uso de la herramienta de sistema make. La razón de ello se encuentra basada en la compatibilidad y funcionalidad de los programas desarrollados en cualquier distribución existente.

A modo de resumen, la herramienta make permite compilar código siguiendo unas directivas marcadas en un archivo denominado Makefile, puesto que no se especifican comandos, sino llamadas a funcionalidades del sistema, se evitan incompatibilidades entre distribuciones o versiones de la misma distribución.

Antes de comenzar con la exposición se debe aclarar que el uso de Makefiles no es la única forma de compilación existente en Linux, aunque si una de las más usadas, pudiendo encontrarse otras opciones:

- Compilación básica por línea de comandos (Alcanza niveles de dificultad desmesurados cuando se deben compilar más de dos o tres archivos). Su ejecución se basa en una llamada al compilador a usar seguida de una serie de opciones y concluyendo con una referencia a los archivos de entrada y salida. Ejemplo:

  gcc -o nombre_archivo_salida Entrada_a_compilar.c
  Otro_Archivo_a_compilar.c -I.

- Uso de CmakeFiles. Para muchos, esta herramienta es la evolución natural de los makeFiles. Puesto que su sintaxis y empleo es similar a la de los MakeFiles, presentando ciertas simplificaciones y aclaraciones, se ha optado por contemplar únicamente una guía de elaboración de esta primera herramienta, ya que una vez conocidos los conceptos de la misma resulta relativamente sencillo el empleo de “su evolución” : CMake.

- Entornos de desarrollo con soporte para Linux, como QtCreator o Eclipse. Cada vez es más frecuente el uso de entornos de desarrollo que, una vez realizada su configuración de forma correcta, generan automáticamente los MakeFiles y compilan el código desarrollado.

De entre todas estas herramientas, a lo largo del presente anexo, será presentada la herramienta make. De acuerdo a lo señalado en las prácticas de informática de [M Ibáñez] [23] ; la herramienta make es utilizada habitualmente en el desarrollo de proyectos con gran número de archivos interdependientes. Esta herramienta utiliza reglas definidas en un archivo llamado 'makefile' para construir un archivo 'destino' especificado por el usuario a partir de un cierto número de archivos 'fuente'. El archivo destino se crea únicamente si no existe o está desactualizado, esto es, si alguno de los archivos fuente es más reciente que el archivo destino. Dicho de otra forma, el archivo destino es construido nuevamente si alguno de los archivos fuente, de los que depende, ha sido modificado desde la última vez que el archivo destino fue creado. Esto implica que al construir un proyecto completo constituido por múltiples archivos, sólo aquellos desactualizados volverán a ser construidos, ahorrando de este modo una cantidad considerable de tiempo y recursos.

Make presenta la siguiente estructura de llamada:

*make [opciones] [destino(s)]*

Las opciones más habituales son:

- -h: Muestra ayuda sobre make.
- -f archivo: Indica que el archivo que contiene las reglas no se llama 'makefile' ni 'Makefile', sino 'archivo'.
- -n: Muestra los comandos que make ejecutaría, pero sin ejecutarlos realmente. Útil para testear makefiles.

23 [M. Ibáñez] Laboratorio de Arquitectura de Ordenadores. Práctica 1: Makefile. Mario Ibañez. Departamento de Ingeniería Telemática. Universidad Carlos III, Madrid.

### 2.3.1. Compilación básica de C en Linux: Gcc y G++.

En este apartado se abordará brevemente el uso de estos dos comandos de compilación en terminal dado que el proceso de elaboración de Makefiles precisa de ellos. La principal diferencia entre ellos radica únicamente en el tipo de compilador empleado; por ello, se citarán únicamente los aspectos de GCC (el uso de G++ es análogo). GCC, puede definirse, según [Victor A. González] [24], como un compilador integrado del proyecto GNU para C, C++, Objective C y Fortran; capaz de recibir un programa fuente en cualquiera de estos lenguajes y generar un programa ejecutable binario en el lenguaje de la máquina donde ha de correr.

La sigla GCC significa "GNU Compiler Collection". Originalmente significaba "GNU C Compiler". No obstante, todavía se usa GCC para designar una compilación en C, recurriéndose a G++ para hacer referencia a una compilación en C++.

*Sintaxis.*

*gcc [ opción | archivo ] ...*

*g++ [ opción | archivo ] ...*

Las opciones van precedidas de un guion, como es habitual en UNIX, pudiendo emplearse un compendio de las mismas. Además, pueden darse varios nombres de archivo a incluir en el proceso de compilación.

Ejemplos.

*gcc hola.c*

*Compila el programa en C hola.c, generando un archivo ejecutable a.out.*

*gcc -o hola hola.c*

*Compila el programa en C hola.c, generando un archivo ejecutable hola.*

---

*24 [VICTOR A.GONZÁLEZ] HTTP://IIE.FING.EDU.UY/~VAGONBAR/GCC-MAKE/GCC.HTM VÍCTOR A. GONZÁLEZ BARBONE INSTITUTO DE INGENIERÍA ELÉCTRICA - FACULTAD DE INGENIERÍA - MONTEVIDEO, URUGUAY.*

*g++ -o hola hola.cpp*

*Compila el programa en C++ hola.c, generando un archivo ejecutable hola.*

*gcc -c hola.c*

No genera el ejecutable, sino el código objeto, en el archivo hola.o. Si no se indica un nombre para el archivo objeto, se emplea el nombre del archivo en C y cambia la extensión por .o.

*gcc -c -o objeto.o hola.c*

Genera el código objeto indicando el nombre de archivo.

*gcc -o ~/bin/hola hola.cpp*

Genera el ejecutable hola en el subdirectorio bin del directorio propio del usuario.

*gcc -L/lib -L/usr/lib hola.cpp*

Indica dos directorios donde han de buscarse bibliotecas. La opción -L debe repetirse para cada directorio de búsqueda de bibliotecas.

*gcc -I/usr/include hola.cpp*

Indica un directorio para buscar archivos de encabezado (de extensión .h).

**Opciones Posibles.**

- c

Realiza preprocesamiento y compilación, obteniendo el archivo en código objeto; no realiza el enlazado.

- E

Realiza solamente el preprocesamiento, enviando el resultado a la salida estándar.

-o archivo

Indica el nombre del archivo de salida, cualesquiera sean las etapas cumplidas.

-Iruta

Especifica la ruta hacia el directorio donde se encuentran los archivos marcados para ser incluidos en el programa fuente. No lleva espacio entre la I y la ruta, así: -I/usr/include

-L

Especifica la ruta hacia el directorio donde se encuentran los archivos de biblioteca con el código objeto de las funciones referenciadas en el programa fuente. No lleva espacio entre la L y la ruta, así: -L/usr/lib

-Wall

Muestra todos los mensajes de error y advertencia del compilador, incluso algunos cuestionables pero en definitiva fáciles de evitar escribiendo el código con cuidado.

-g

Incluye en el ejecutable generado la información necesaria para poder rastrear los errores usando un depurador, tal como GDB (GNU Debugger).

-v

Muestra los comandos ejecutados en cada etapa de compilación y la versión del compilador. Es un informe muy detallado.

### 2.3.2. Partes de un Makefile.

Tal y como se ha señalado, y según añade [G Aburruzaga] [25]. La información que necesita make para recompilar un sistema se obtiene mediante la lectura y análisis de archivo llamado Makefile. Pudiendo clasificarse el contenido de un Makefile como:

- **Reglas**. Una regla señala cuándo y cómo se debe uno o más ficheros, denominados "objetivos de la regla". En ella se listan, en primer

25[G. Aburrazaga] Make. Un programa para controlar la recompilación. Gerardo Aburruzaga García.

lugar, los objetivos y separados por el signo dos puntos, los otros ficheros de los cuales depende el objetivo, llamados "dependencias"; y, por último, en las siguientes líneas, después un carácter tab (tabulador), se puede dar una serie de órdenes para efectuar la tarea. A veces los objetivos y las dependencias no son ficheros reales sino que hacen referencia a sub-reglas.

- **Macros**. Una *definición de variable o macro* se refiere una línea que especifica un valor textual para una variable o macro; este valor podrá ser sustituido posteriormente las veces que haga falta.
- **Comentarios**. Un comentario es aquel texto que va desde el signo '#' hasta el final de la línea. No obstante, este signo podría variar ya que en las líneas de órdenes (recordar que los Makefiles se ejecutan como si de órdenes introducidas por consola se tratase), es el Shell o intérprete de órdenes el que decide qué es un comentario, sin embargo, suele coincidir con este signo. Por supuesto, make no los tiene en cuenta, así como las líneas en blanco tampoco son consideradas.
- **Directivas**. Una directiva o directriz es una orden para make que le insta a realizar una acción especial mientras está leyendo el makefile, como por ejemplo, incluir otro en este. Las directivas dependen mucho de la versión de make que se esté usando, y su uso no será abordado en este anexo.

### 2.3.3. Proceso de creación de un Makefile.

Generalmente se otorga el nombre Makefile al archivo que deseamos emplee la herramienta make para compilar. No obstante, es posible emplear otros nombres si al ejecutar el comando make se emplea el modificador –f, otorgándose como argumento secundario el nombre del archivo a buscar. Así mismo, es frecuente crear este archivo de compilación en el mismo directorio que los ficheros fuentes o, en su defecto, en el directorio anterior en caso de que éstos se encuentren en la carpeta src. De las definiciones realizadas en el apartado precedente se extrae que las reglas constituyen el contenido básico y esencial de todo Makefile.

Una regla aparece en el Makefile y describe cuándo y cómo reconstruir ciertos ficheros, llamados objetivos de la regla (normalmente uno por cada regla). A continuación lista las dependencias del objetivo, y las órdenes necesarias para crearlo o actualizarlo.

El orden de las reglas no tiene importancia salvo para determinar la predeterminada, es decir, el objetivo que make construirá, si no se especifica otro en la línea de órdenes. Esta regla predeterminada es la primera que aparece en el makefile. Por lo tanto, se debe escribir el makefile de forma que la primera regla que se ponga sea la encargada de compilar el programa entero, o todos los programas que se desee compilar por defecto.

En general, una regla aparece en un Makefile con una sintaxis de la forma:

*objetivo(s): dependencia(s)*

*tab orden*

*tab orden*

*. . .*

Donde:

**Objetivos.** Son nombres de ficheros o palabras, separadas por espacios en blanco. Se pueden usar comodines. Normalmente sólo se especifica un objetivo, pero, en ciertas ocasiones, es necesario emplear emplear multiobjetivos.

**Dependencias.** Son nombres de ficheros u otros objetivos, de los que dependen, separados por espacios en blanco. Se pueden usar comodines.

**Órdenes o comandos.** Comandos que serán interpretados por el sistema operativo, y que normalmente se utilizan para construir el destino, aunque se puede utilizar cualquier tipo de comando. Es muy importante recordar que cada comando debe ir precedido de un tabulador, de otro modo make produciría un error. Para dividir un comando en varias líneas, se puede utilizar el carácter '\' tras cada línea. Un ejemplo típico de comando sería 'gcc -c data.c'.

Todos estos conceptos se recogen en el siguiente ejemplo de Mario Ibáñez: supóngase que se tiene una aplicación capaz de planificar un viaje calculando el tiempo que lleva conducir de una ciudad a otra a una velocidad dada. Encontrándose esta aplicación compuesta por los siguientes archivos:

cities.txt: archivo en el que el usuario introduce las ciudades que quiere incluir en la aplicación, en unalista de texto.

- **dist_table.dat:** archivo que contiene una tabla con todas las ciudades y las distancias entre ellas.
- **create_table.c:** código fuente para un programa que comprueba que todas las ciudades incluidas en cities.txt están incluidas en la tabla de 'dist_table.dat', si alguna falta pide las distancias desde ella al resto de ciudades y las incluye en la tabla.
- **travel_time.c:** código fuente para un programa que calcula el tiempo necesario para viajar entre dos ciudades a una velocidad dada en km/h (por ejemplo 'travel_time Madrid Santander 120').
- **input.h:** archivo de cabecera con funciones para leer de un archivo, incluido por ambos archivos de código fuente.
- **foutput.h:** archivo de cabecera con funciones para escribir en un archivo, incluido por create_table.c.
- **scr_io.h:** archivo de cabecera con funciones para leer o escribir por pantalla en un archivo, incluido por ambos archivos de código fuente.

Siendo el árbol de dependencias:

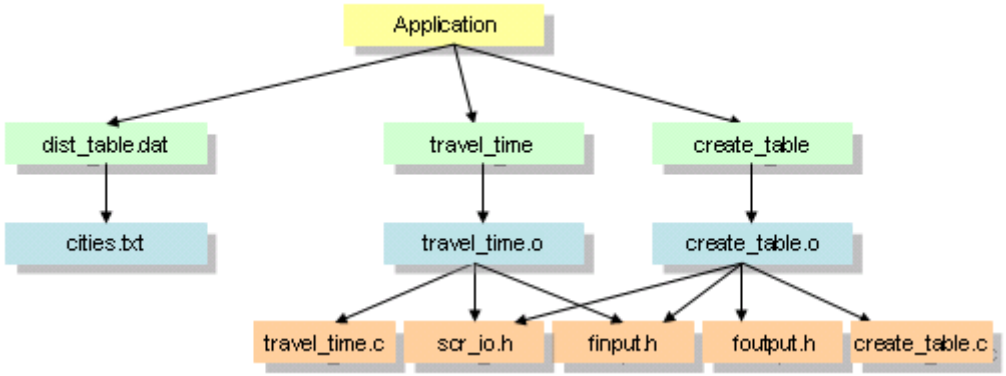



*Ilustración 46. Arbol de dependencias del primer ejemplo de MakeFile.*

De forma que el MakeFile a usar para la compilación sería:

```
# Makefile para la aplicación de cálculo del tiempo de viaje

travel_time: travel_time.o
   gcc -o travel_time travel_time.o

create_table: create_table.o
   gcc -o create_table create_table.o

travel_time.o: travel_time.c scr_io.h finput.h
   gcc -c travel_time.c

create_table.o: create_table.c scr_io.h finput.h foutput.h
   gcc -c create_table.c

dist_table.dat: cities.txt
   create_table cities.txt
```

Según la exposición realizada previamente, de la observación del MakeFile se obtiene lo siguiente:

- travel_time es la aplicación global a generar, para cuya compilación en c (compilador gcc) se necesita del objeto travel_time.o
- La construcción del archivo travel_time.o que, para su compilación, requiere los archivos travel_time.c, scr_io.h finput.h.
- Con create_table acontece un proceso análogo.
- La última directiva permite generar el fichero en el que se almacenarán todos los datos.

De este ejemplo se puede extraer que para la creación de Makefiles resulta imperativo tener nociones de los comandos empleados para la compilación en terminal: gcc y g++, mediante los cuales es posible generar ejecutables en c y c++ respectivamente.

No obstante, el Makefile visto anteriormente carece de comandos de ejecución automática, hecho que implica la escritura de tres variantes de make para compilar todo el programa:

```
> make create_table
> make travel_time
> make dist_table.dat
```

Para solventar este inconveniente es preciso emplear los denominados destinos simbólicos:

```
# Makefile for travel time application with a macro
DATA = dist_table.dat
all: travel_time create_table $(DATA)

travel_time: travel_time.o
   gcc -o travel_time travel_time.o

create_table: create_table.o
   gcc -o create_table create_table.o

travel_time.o: travel_time.c scr_io.h finput.h
   gcc -c travel_time.c

create_table.o: create_table.c scr_io.h finput.h foutput.h
   gcc -c create_table.c

$(DATA): cities.txt
   create_table cities.txt

clean:
   rm *.o
```

En el ejemplo se ha incluido un segundo destino simbólico llamado 'clean', que carece de lista de dependencias, con lo que los comandos se ejecutan únicamente si el destino es invocado de forma directa en la llamada a make. Con este makefile, construir la aplicación completa se reduce a teclear 'make all'. Además, de no especificarse ningún destino, make asume que sólo se desea construir el primero, de modo que se podría utilizar simplemente 'make'. Por otro lado, para borrar todos los archivos objeto, se ejecutaría 'make clean.

Para concluir, se ejemplificará el uso de macros para sustituir el texto y obtener un archivo más ordenado y estructurado. Para sustituir una subcadena correspondiente a una macro por la cadena que ésta "almacena", se puede emplear la sintaxis $(MACRO). En caso de requerir modificar la cadena almacenada por la macro se recurre a la sintaixs $(MACRO:antigua_subcadena=nueva_subcadena). Como se puede comprobar, esta herramienta es aprovechada en el ejemplo para utilizar la misma macro para el nombre del ejecutable, el archivo de código fuente y el archivo objeto. También existe la posibilidad de definir una macro en la llamada a make, especificando 'make NOMBRE_MACRO=valor'. Así, en este ejemplo, la macro DATA se define llamando a make con 'make DATA=dist_table.dat'

```
# Makefile para la aplicación de cálculo del tiempo de viaje con
# Sustitución de texto en macros
PROG1 = travel_time
PROG2 = create_table

all: $(PROG1) $(PROG2) $(DATA)
$(PROG1): $(PROG1).o
  gcc -o $(PROG1) $(PROG1).o

$(PROG2): $(PROG2).o
  gcc -o $(PROG2) $(PROG2).o

$(PROG1).o: $(PROG1).c scr_io.h finput.h
  gcc -c $(PROG1).c

$(PROG2).o: $(PROG2).c scr_io.h finput.h foutput.h
  gcc -c $(PROG2).c
```

```
$(DATA): cities.txt
  $(PROG2) cities.txt

clean:
  rm *.o
```

## 2.4. Relación de Comandos de Utilidad.

A continuación, se mostrará una lista con los principales comandos que es necesario conocer para interactuar de forma básica con el sistema a través del terminal:

### 2.4.1. Manuales de Comandos.

- **man → (manual: manual)**

Nos ofrece el manual de cualquier comando en la propia terminal. Para utilizarlo, basta con ejecutar "man" seguido del comando del que se desea saber más o simplemente recordar:

*man comando*

En ocasiones la información que ofrece man puede llegar a ser excesiva. Casi todos los comandos y aplicaciones aceptan el argumento *"--help" o "-h"* para que muestre cierta ayuda más resumida. Por ejemplo con "apt-get":

*apt-get --help*

*o*

*apt-get -h*

### 2.4.2. Comandos Relacionados Con Archivos y Directorios.

- **ls → (list: listar)**

Muestra el contenido de la carpeta que indiquemos después.

La sinapsis del comando sería:

ls [opciones] [ruta]

Opciones:

-a → Muestra todos los ficheros incluyendo algunos que ordinariamente están ocultos para el usuario (aquellos que comienzan por un punto). Se debe recordar que el fichero punto. Indica el directorio actual y el doble punto .. el directorio padre, que contiene, al actual.

-l → Esta es la opción de lista larga: muestra toda la información de cada fichero incluyendo: protecciones, tamaño y fecha de creación o del último ambio introducido,...

-c → Muestra ordenando por día y hora de creación.

-t → Muestra ordenando por día y hora de modificación.

-r → Muestra el directorio y lo ordena en orden inverso.

-R → Lista también subdirectorios.

ls subdir → Muestra el contenido del subdirectorio subdir.

-l filename → Muestra toda la información sobre el fichero filename.

--color → Muestra el contenido del directorio coloreado.

Ejemplos:

Si se quiere que muestre lo que contiene el directorio o carpeta "/etc":

*ls /etc*

Si no se pone nada interpretará que lo que se quiere ver es el contenido de la carpeta donde se encuentra actualmente:

*Ls*

- **file**

Este comando realiza una serie de comprobaciones en un fichero para tratar de clasificarlo, mostrando sus características.

La sinapsis del comando sería:

file [OPCIÓN...] [ARCHIVO...]

Tras su ejecución este comando muestra el tipo del fichero e información al respecto del mismo. Este comando se puede aplicar también a directorios.

- **cd → (change directory: cambiar directorio)**

Se utiliza para cambiar de directorio o carpeta en la terminal. Se usar usar con rutas absolutas o relativas. En las absolutas se indica toda la ruta desde la raíz (/). Por ejemplo, al escribir en consola…

*cd /etc/apt*

nos llevará a esa carpeta directamente.

Las rutas relativas son relativas a algo, y ese algo es la carpeta donde se encuentre actualmente. Si se supone que el sistema se encuentra en /home y se desea ir a la carpeta "Imágenes" dentro de vuestra carpeta personal. Con escribir

*cd Imágenes*

bastará. Como se puede observar se ha obviado el "/home/carpeta_personal" inicial ya que de no ser introducida se considera como referencia el directorio en el que se haya en el momento de ejecutar el.

➢ **mkdir → (make directory: hacer directorio)**

Crea una carpeta o directorio con el nombre que se indique. Siendo posible usar rutas absolutas y relativas. Se puede indicar toda la ruta que precede al directorio que se quiera crear:

*mkdir /home/carpeta_personal/nueva_carpeta*

O de estar ya en la carpeta que lo va a contener, basta con poner tan sólo el nombre de la nueva carpeta. Por ej. si se encuentra en /home/carpeta_personal:

*mkdir nueva_carpeta*

➢ **rm → (remove: borrar)**

Borra el archivo o la carpeta que se indique. Al igual que con otros comandos, se puede indicar la ruta completa o el nombre del archivo.

Para borrar un archivo:

*rm nombre_archivo*

Para borrar un directorio o carpeta vacía:

*rm nombre_carpeta*

Para **borrar un directorio o carpeta que contiene archivos** y/o otras carpetas que pueden, a su vez, contener más carpetas y archivos:

rm -r nombre_carpeta

Otras opciones:

"-f", no pide una confirmación para eliminar.

"-v", va mostrando lo que va borrando.

➢ **cp → (copy: copiar)**

Copia el archivo o directorio indicado donde se especifique. Admite ambos tipos de nomenclatura de rutas, tanto para el fichero origen, como en el del destino.

La sinapsis del comando sería:

cp [/ruta/de/original...] [/ruta/de/copia...]

Por ejemplo, en nuestra carpeta personal se va a crear una copia de seguridad "sources.list.backup", de los repositorios en "/etc/apt/sources.list". Este ejemplo será explicado según variando la localización en el terminal, para comprender lo primordial que es saber en todo momento el directorio en el que uno se encuentra trabajando:

- Si se está colocado en la carpeta personal, se debe de poner la ruta absoluta del original y la ruta relativa de la copia:

*cp /etc/apt/sources.list sources.list.backup*

- Si por el contrario, se encuentra colocado en el directorio que contiene el archivo original (cd /etc/apt), se debe de poner la ruta relativa del original y la ruta absoluta de la copia:

*cp sources.list /home/tu_usuario/sources.list.backup*

ó

*cp sources.list ~/sources.list.backup*

- Si se estuviese en cualquier otro directorio o simplemente para no tener problemas, se escriben las dos rutas absolutas:

cp /etc/apt/sources.list /home/tu_usuario/sources.list.backup

➢ **mv → (move: mover)**

Es igual que el anterior, sólo que en lugar de hacer una copia, mueve directamente el archivo con el nombre que se indique, pudiendo ser otro distinto al original:

La sinapsis del comando sería idéntica a copiar:

mv [/ruta/de/original...] [/ruta/de/destino...]

Otro uso muy práctico que se le puede dar es para renombrar un archivo. Basta con indicar el nuevo nombre en el segundo argumento con la misma ruta del primero. En este ejemplo se supone que ya se encuentra en la carpeta que lo contiene:

*mv archivo.flv mi_archivo.flv*

➢ **pwd → (print working directory)**

Visualiza o imprime la ruta del directorio en el que se encuentra en ese preciso momento. Este comando es uno de los pocos que no tiene opciones y se utiliza escribiendo simplemente:

*Pwd*

➢ **find → (find: encontrar)**

Busca archivos o carpetas en la ruta que le indiques:

La sinapsis del comando sería:

find [/directorio/donde/buscar...] [-expresión] [búsqueda]

Donde "expresión" es el tipo de búsqueda y siempre se le antepone el signo "-"

La expresión "-name" sería para realizar una búsqueda por nombre. Por ejemplo, para buscar en todo el sistema de archivos o raíz "/" las carpetas y archivos que se llamen "busqueda". Sería:

*find / -name busqueda*

Si se tuviese la seguridad de que se encuentra en /var por ejemplo, se le indicaría de la forma:

*find /var -name busqueda*

Si se tienen dudas del nombre es posible indicarlo con comodines. Suponga que el nombre de lo que buscamos contiene “bus”, en la misma carpeta de antes:

*find /var -name *bus**

Otra expresión sería "-size" para realizar la búsqueda por tamaño. Por ejemplo se puede solicitar que encuentre los archivos/carpetas de más de 1500 KB:

*find / -size +1500*

Se pueden combinar varios atributos para afinar la búsqueda. Por ejemplo, buscar los archivos/carpetas que contienen el nombre “bus” y tienen menos de 1000 KB:

*find / -name *bus* -size -1000*

La opción "2>/dev/null" es muy interesante para que no muestre los errores de "Permiso denegado". Por ejemplo para buscar en la raíz "/" el archivo "gdmflexiserver":

find / -name gdmflexiserver 2>/dev/null

- **grep → (localizar)**

El comando grep localiza una palabra, clave o frase en un conjunto de directorios, indicando en cuáles de ellos la ha encontrado. Este comando rastrea fichero por fichero, por turnos, imprimiendo aquellas líneas que contienen el conjunto de caracteres buscado. Si el conjunto de caracteres a buscar está compuesto por dos o más palabras separadas por un espacio, se colocará el conjunto de caracteres entre apóstrofes (').

La sinapsis del comando sería:

*grep [OPCIÓN] 'conjuntocaracteres' [ARCHIVOS...]*

siendo 'conjuntocaracteres' la secuencia de caracteres a buscar, y file1, file2, y file3 los ficheros donde se debe buscar.

Las opciones principales del comando son:

-c → lo único que se hace es escribir el número de las líneas que satisfacen la condición.

-i → no se distinguen mayúsculas y minúsculas.

-l → se escriben los nombres de los ficheros que contienen líneas buscadas.

-n → cada línea es precedida por su número en el fichero.

-s → no se vuelcan los mensajes que indican que un fichero no se puede abrir.

-v → se muestran sólo las líneas que no satisfacen el criterio de selección.

A continuación se muestra una serie de ejemplos.

grep '^d' text → líneas que comienzan por d.

grep '^[^d]' text → líneas que no comienzan por d.

grep -v '^C' file1 > file2 → quita las líneas de file1 que comienzan por C y lo copia en file2.

➢ **cat → (Visualización sin formato de un fichero)**

Este comando permite visualizar el contenido de uno o más ficheros de forma no formateada. También permite copiar uno o más ficheros como apéndice de otro ya existente. Algunas formas de utilizar este comando son las siguientes:

Sacar por pantalla el contenido del fichero filename:

*cat filename*

Mostrar por pantalla, secuencialmente y según el orden especificado, el contenido de los ficheros indicados (file1 y file2):

*cat file1 file2*

Aceptar lo que se introduce por teclado y almacenarlo en file1 (se crea file1):

*cat >file1*

### 2.4.3. Comandos Relacionados con Sistema y Administración.

- **ps → (process status: estado de los procesos)**

Muestra información sobre los procesos que están siendo ejecutados en el sistema. Cada proceso está identificado con un número llamado PID. Si se escribe

*ps -A*

se mostrará un listado de todos los procesos, su PID a la izquierda y su nombre a la derecha. En caso de querer más información:

*ps aux*

- **kill → (kill: matar)**

Permite enviar señales a uno o varios procesos del sistema. Las más utilizadas suelen ser la de matar un proceso (9 o SIGKILL), detener el proceso (TERM) o reiniciarlo (1 o HUP), sin embargo, hay muchas más que pueden resultar útiles. El listado completo de señales disponibles puede visualizarse ejecutando:

*kill -l*

Como señal se puede utilizar el número correspondiente a la izquierda del nombre de la señal (SIG...) o escribir directamente el nombre sin el "SIG" que le precede, por ejemplo "STOP"

La sinapsis del comando sería:

kill [señal] <pid> [...]

Por ejemplo, para solicitar que termine un proceso cuyo PID es "3760" basta con ejecutar kill seguido del PID correspondiente. Para esta llamada se está empleado la señal SIGTERM(15), que es la asociada por defecto, por ello no se refleja en la nomenclatura:

*kill 3760*

Para forzar que uno o varios procesos terminen de forma inmediata (sin solicitar ni preguntar…) usamos la señal SIGKILL (9). Esta señal debe ejecutarse con cuidado ya que fuerza a los procesos a terminar inmediatamente sin permitirles terminar de forma limpia, es decir, puede que no borre los PID, que no deje terminar las peticiones pendientes, etc:

*kill -9 3760*

Si se desea forzar a todos los procesos, con un determinado nombre, a que finalicen inmediatamente usaríamos "killall" en lugar de kill. Por ejemplo para cerrar varios programa_de_prueba's que existiesen en el escritorio:

*killall -9 programa_de_prueba*

Otro ejemplo sería el de suspender un proceso, para ello se envía la señal de STOP (19) seguida del proceso. Si el ID de la señal es desconocido, es posible usarla mediante el nombre. En esta señal el proceso quedaría suspendido, por lo que todavía figuraría en la lista de procesos y se podría ser reanudado posteriormete (próximo ejemplo):

*kill -19 3760*

o

*kill -STOP 3760*

Una vez conocida la forma de suspender un proceso, es interesante conocer como reactivarlos, para ello se emplea la señal CONT (18). En el siguiente ejemplo se va a "revivir" el proceso suspendido anterior:

*kill -18 3760*

o

*kill -CONT 3760*

Una de las señales más importantes es HUP (1). Esta señal permite parar y reiniciar el proceso indicado, también se puede aplicar con el nombre del proceso además del ID.

*kill -HUP 3760*

o con el nombre del proceso:

*killall -HUP script.sh*

En caso de querer utilizarlo para, por ejemplo, reiniciar todos los procesos " programa_de_prueba " usaríamos killall en lugar de kill:

killall -HUP *programa_de_prueba*

- **sudo → (super-user do: hacer como superusuario)**

Permite a los usuarios ejecutar acciones con los privilegios de seguridad del root, de manera segura.

Por defecto Ubuntu tiene desactivada la cuenta del "root", por seguridad y para administrar el sistema existe un grupo de usuarios denominado "sudoers users" (administradores o admin), los cuales pueden obtener permisos de root, mediante la utilización de "sudo". Generalmente, el usuario con el que instalamos Ubuntu, se encuentra incluido en este grupo de administradores. En el terminal se utiliza el comando "sudo", anteponiéndolo a la orden o comando a ejecutar:

*sudo orden*

- **su → (switch user: cambio de usuario)**

Cambiar de usuario sin necesidad de hacer un cierre o cambio de sesión:

*su nombreusuario*

La contraseña que nos pedirá, es la del usuario al que vamos a cambiar, no la del usuario en el que estamos.

- **passwd → (password: contraseña)**

Cambia las contraseñas de cuentas de usuario. Los usuarios normales sólo pueden cambiar la contraseña de su propia cuenta y el superusuario puede cambiar todas.

La sinapsis del comando sería:

passwd [opciones] [USUARIO]

Opciones:

-a, --all → informa del estado de las contraseñas de todas las cuentas

-d, --delete → borra la contraseña para la cuenta indicada

-e, --expire → fuerza a que la contraseña de la cuenta caduque

-h, --help → muestra un mensaje de ayuda y termina.

-k, --keep-tokens → cambia la contraseña sólo si ha caducado.

-i, --inactive INACTIVO → establece la contraseña inactiva después de caducar a INACTIVO

-l, --lock → bloquea la contraseña de la cuenta indicada

-n, --mindays DÍAS_MIN → establece el número mínimo de días antes de que se cambie la contraseña a DÍAS_MIN

-q, --quiet → modo silencioso

-r, --repository REP → cambia la contraseña en el repositorio REP

-R, --root CHROOT_DIR → directory to chroot into

-S, --status → informa del estado de la contraseña la cuenta indicada

-u, --unlock → desbloquea la contraseña de la cuenta indicada

-w, --warndays DÍAS_AVISO → establece el aviso de caducidad a DÍAS_AVISO

-x, --maxdays DÍAS_MAX → establece el número máximo de días antes de cambiar la contraseña a DÍAS_MAX

Si se especifica nombre-usuario, se cambiará la contraseña de dicho usuario (para esto se debe ser root), sino, la del usuario que ejecuta el comando. La mecánica de cambio de contraseña tiene 3 pasos:

- Ingresar la contraseña antigua.

- Ingresar la contraseña nueva.

- Repetir la contraseña nueva para confirmar.

➢ **apt → (advanced packets tool: herramienta avanzada de paquetes)**

apt-get es la herramienta que utiliza Debian y sus derivados (Ubuntu incluido), para gestionar los paquetes instalables disponibles en los repositorios.

➢ **dpkg → (depackage: despaquetar)**

Los paquetes cuando se instalan sufren un proceso de desempaquetado. En el fondo, un paquete .deb contiene una serie de scripts de pre-instalación, post-instalación y los archivos en cuestión del paquete.

Este comando se usará para instalar un paquete .deb descargado en nuestro sistema. En muchas ocasiones, hay una aplicación que no está en los repositorios y es necesario descargarla de internet, obteniendo el pertinente .deb que permite instalarlo con el interfaz gráfico que corresponda (GDebi en el caso de GNOME).

Si se quiere instalar un paquete ya descargado mediante consola se usará el argumento '-i' (i=install):

*dpkg -i nombre_paquete*

Para desinstalarlo '-r' (r=remove):

*dpkg -r nombre_paquete*

Para desinstalar el paquete y los ficheros de configuración "–purge" (purgar):

*dpkg -r –purge nombre_paquete*

➢ **date → (date: fecha)**

Muestra por pantalla el día y la hora, permitiendo, además, el cambio de la misma.

La sinapsis del comando sería:

date [OPCIÓN]... [+FORMATO]

o bien:

date [-u|--utc|--universal] [MMDDhhmm[[SS]AA][.ss]]

- **cal → (calender: calendario)**

  Muestra el calendario del mes o año actual actual.

  La sinapsis del comando sería:

  cal [mes] [año]

- **who → (who: quien)**

  Indica qué usuarios tiene el ordenador en ese momento, en qué terminal (tty) está y a qué hora iniciaron la sesión.

  La sinapsis del comando sería:

  who [OPCIÓN]...

- **whoami → (who I am: quien soy)**

  Indica el usuario que está trabajando en la terminal actual.

  La sinapsis del comando sería:

  Whoami

- **finger**

  Presenta una información completa de los usuarios conectados a la red.

  La sinapsis del comando sería:

  finger [-lmsp] [user ...] [user@host ...]

- **uname**

  Proporciona el nombre del sistema en el que se está trabajando.

  La sinapsis del comando sería:

  uname [-opciones]

  Como opciones principales se tienen:

  -a → indica, además, la versión, fecha y tipo de procesador.

  -m → indica, además, el tipo de de procesador.

  -r → indica, además, la versión.

  -v → indica, además, la fecha.

- **clear**

  Este comando se utiliza para limpiar la pantalla de la terminal.

### 2.4.4. Caracteres Comodín o Wildcards.

Una característica importante de la mayoría de los intérpretes de comandos en Linux es la capacidad para referirse a más de un fichero. Una forma de hacerlo es utilizando caracteres especiales llamados comodines.

Al igual que en MS-DOS, el comodín * hace referencia a cualquier carácter o cadena de caracteres presente en el nombre del fichero. El intérprete de comandos sustituirá el asterisco por todas las combinaciones posibles provenientes de los ficheros en el directorio al cual nos estamos refiriendo. En ocasiones, se dice que está realizando una expansión de comodines.

El carácter ? es también comodín, aunque solamente expande un carácter. Con ambos caracteres existe una excepción. *No afectarán a aquellos ficheros que comienzan por un punto, y que son ocultos para órdenes como ls.*

Además, podemos utilizar los corchetes para referirnos a un conjunto de caracteres o bien un rango de caracteres ASCII.

Ejemplos:

- ls *n* → muestra todos los archivos y directorios, del directorio actual, que contienen el carácter n
- ls * → muestra todos los archivos y directorios del directorio actual
- ls tm? → muestra todos los archivos y directorios del directorio actual que comienzan por tm y contienen tres caracteres
- ls tabla[123]a → muestra todos los archivos y directorios del directorio actual que comienzan por tabla, seguidos del carácter 1, 2 ó 3, y terminan en a
- ls ??base[A-Z][5-9]* → muestra todos los archivos y directorios del directorio actual que comienzan con dos caracteres cualesquiera, seguidos de la cadena base, a continuación una letra mayúscula, seguida de un número del 5 al 9 y por último una cadena de caracteres (uno, varios o ninguno)

### 2.4.5. Agrupación y Compresión de Ficheros: Comandos TAR y GZIP/GUNZIP.

Tanto el comando tar como gzip son ampliamente empleados para la difusión de programas y ficheros en Linux.

➢ **tar**

Este comando agrupa varios ficheros en un solo "archivo", mientras que el segundo los comprime. En conjunto estos dos programas actúan de forma muy similar a programas como Winzip. Su sintaxis es:

tar [opciones][ficheros]

El modo en el que se escriben las opciones de tar es un tanto especial. El guion inicial, por ejemplo, no es necesario.

Las opciones más comunes para tar son:

-c creación de archivadores nuevos.

-x extracción de archivos de un archivador existente.

-v muestra los archivos mientras se agregan o se extraen.

-t muestra el contenido de un archivo tar.

-f el siguiente argumento es el archivador a crear, del que queremos extraer archivos o mostrar un listado.

Para crear un nuevo archivo se emplea:

*tar –cvf nombre_archivo.tar fichero1 fichero2 …*

donde fichero1, fichero2 etc. son los ficheros que se van a añadir al archivo tar. Si se desea extraer los ficheros se emplea:

tar –xpvf nombre_archivo.tar fichero1 ...

- **gzip/gunzip.**

Al contrario que tar, que agrupa varios ficheros en uno, gzip comprime un único fichero con lo que la información se mantiene pero se reduce el tamaño del mismo. El uso de gzip es muy sencillo:

gzip [opciones] fichero

con esta orden se comprime fichero (que es borrado) y se crea un fichero con nombre fichero.gz.

La opción más común es:

-1 a –9 grado de compresión, mínimo y máximo respectivamente.

-d descomprimir el fichero .gz

Si lo que se desea es descomprimir un fichero se emplea entonces:

*gzip –d fichero.gz*

recuperándose el fichero inicial.

Otra posibilidad sería utilizar el comando gunzip para la descompresión, de la siguiente forma:

*gunzip fichero.gz*

Como se ha comentado al principio, es típico emplear tar y gzip de forma consecutiva, para obtener ficheros con extensión tar.gz o tgz que contienen varios ficheros de forma comprimida (similar a un fichero zip). El comando tar incluye la opción z para estos ficheros, de forma que; para extraer los ficheros que contiene:

tar –zxf fichero.tar.gz

### 2.4.6. Cambio de Modo de los Ficheros: CHMOD, CHOWN Y CHGRP.

Cada usuario es dueño de su directorio personal y será dueño también de los archivos que incluya en él. Un usuario en Linux podrá configurar permisos en sus archivos. Por ello, se distingue, por un lado, tres categorías de usuarios, y por otro, los tipos de permisos que cada uno de ellos puede tener sobre un archivo y/o directorio.

Categorías de usuarios

- Dueño del archivo (u).
- Grupo dueño (g), formado por todos los usuarios que son miembros de un grupo asociado al archivo.
- Resto de usuarios (o), todos los usuarios que no son ni el dueño ni miembros del grupo dueño.

Tipos de permisos

- Lectura (r de Read, leer): para un archivo dado, permite leer su contenido. Mientras que para un directorio permite que se muestren los archivos que contiene.
- Escritura (w de Write, escribir): para un archivo dado, permite que se modifique su contenido. Mientras para un directorio permite agregar y quitar archivos.

- Ejecución (x de eXecute, ejecutar): para un archivo dado, permite su ejecución. Mientras que para un directorio permite que el usuario lo recorra (que entre y pase por él) – si no tiene permiso de lectura, aunque pueda entrar no podrá ver el contenido.

Cuando se ejecuta el comando ls –l nombre_archivo, se puede ver la configuración de permisos del archivo nombre_archivo:

El primer carácter indica el tipo de archivo: “d” si es directorio, “-“ si es un archivo regular, “l” si es un enlace simbólico.

Los siguientes nueve caracteres indican los permisos para el dueño, el grupo dueño y otros (rwxrwxrwx); si aparece un guión, indica que el permiso correspondiente no está habilitado.

El siguiente número indica el número de vínculos.

Nombre del dueño y nombre del grupo dueño.

Tamaño en bytes.

Fecha de la última modificación.

Nombre del archivo.

❖ **Comando chmod** -> Para cambiar los permisos de un fichero se emplea el comando chmod, que tiene el formato siguiente:

chmod [quien] oper permiso files

donde:

quien -> Indica a quién afecta el permiso que se desea cambiar. Es una combinación cualquiera de las letras "u" para el usuario, "g" para el grupo del usuario, "o" para los otros usuarios, y "a" para todos los anteriores. Si no se da el quién, el sistema supone "a".

oper -> Indica la operación que se desea hacer con el permiso. Para dar un permiso se pondrá un +, y para quitarlo se pondrá un -. Si quiero dar exactamente unos permisos, pondremos =.

permiso -> Indica el permiso que se quiere dar o quitar. Será una combinación cualquiera de las letras anteriores: r,w,x,s.

files -> Nombres de los ficheros cuyos modos de acceso se quieren cambiar.

Los permisos de lectura, escritura y ejecución tienen un significado diferente cuando se aplican a directorios y no a ficheros normales. En el caso de los directorios el permiso r significa la posibilidad de ver el contenido del directorio con el comando ls; el permiso w da la posibilidad de crear y borrar ficheros en ese directorio, y el permiso x autoriza a buscar y utilizar un fichero concreto.

- **Comando chown** -> Por otra parte, el comando chown se emplea para cambiar de propietario ("change owner") a un determinado conjunto de ficheros. Este comando sólo lo puede emplear el actual propietario de los mismos. Los nombres de propietario que admite Linux son los nombres de usuario, que están almacenados en el fichero /etc/passwd.

  La forma general de utilización del comando chown es:

  *chown newowner file1 file2 …*

- **Comando chgrp** -> Análogamente, el grupo al que pertenece un fichero puede ser cambiado con el comando chgrp, que tiene una forma general similar a la de chown,

  chgrp newgroup file1 file2...

  Los grupos de usuarios están almacenados en el fichero /etc/group.

Esta información se ha recopilado navegando por los distintos foros y posts de la comunidad española de Ubuntu, habiendo sido gran parte de la información aquí presente extraída de http://www.ubuntu-guia.com/2009/07/comandos-basicos-de-linux.html. Existen muchos más comandos. Sin embargo, este apartado trata de servir de mera guía o marcar algunas directrices para facilitar las primeras andaduras que un usuario "novato" realice en Ubuntu. Por ello y dada la amplia y activa comunidad existente, la relación de comandos concluye aquí y con ella, el presente anexo.

# Anexo 3. Relación de robots integrados en ROS satisfactoriamente.

A continuación se muestra una relación de los robots integrados en ROS con plena funcionalidad, habiendo sido ésta obtenida del sitio web oficial [26] :

- Fraunhofer IPA Care-O-bot
- Videre Erratic
- Turtlebot
- Nao
- Aldebaran Nao
- Lego NXT
- Shadow Hand
- Willow Garage PR2
- iRobot Roomba
- Robotnik Guardian
- Merlin miabotPro
- AscTec Quadrotor
- CoroWare Corobot
- Husky
- Clearpath Robotics Kingfisher
- Clearpath Robotics Grizzly
- Gostai Jazz
- Neobotix mpo-500
- Neobotix mpo-700
- Modular Arm
- Robotnik Summit

[26] HTTP://WIKI.ROS.ORG/ROBOTS

- Robonaut 2
- Adept MobileRobots Pioneer family (P3DX, P3AT, PeopleBot, PowerBot, AmigoBot, PatrolBot, GuiaBot
- Seekur and compatible
- Adept MobileRobots Pioneer LX
- AMIGO
- Allegro Hand
- Komodo
- WheeledRobin
- Kawada Nextage
- Denso VS060
- Kinova JACO
- Otto Bock SensorHand Speed
- Robotino
- Robotnik Agvs
- Kinova MICO
- BarrettHand

Esta lista se encuentra ordenada según el orden de integración tal y como se cita en la fuente de la que se ha extraido.

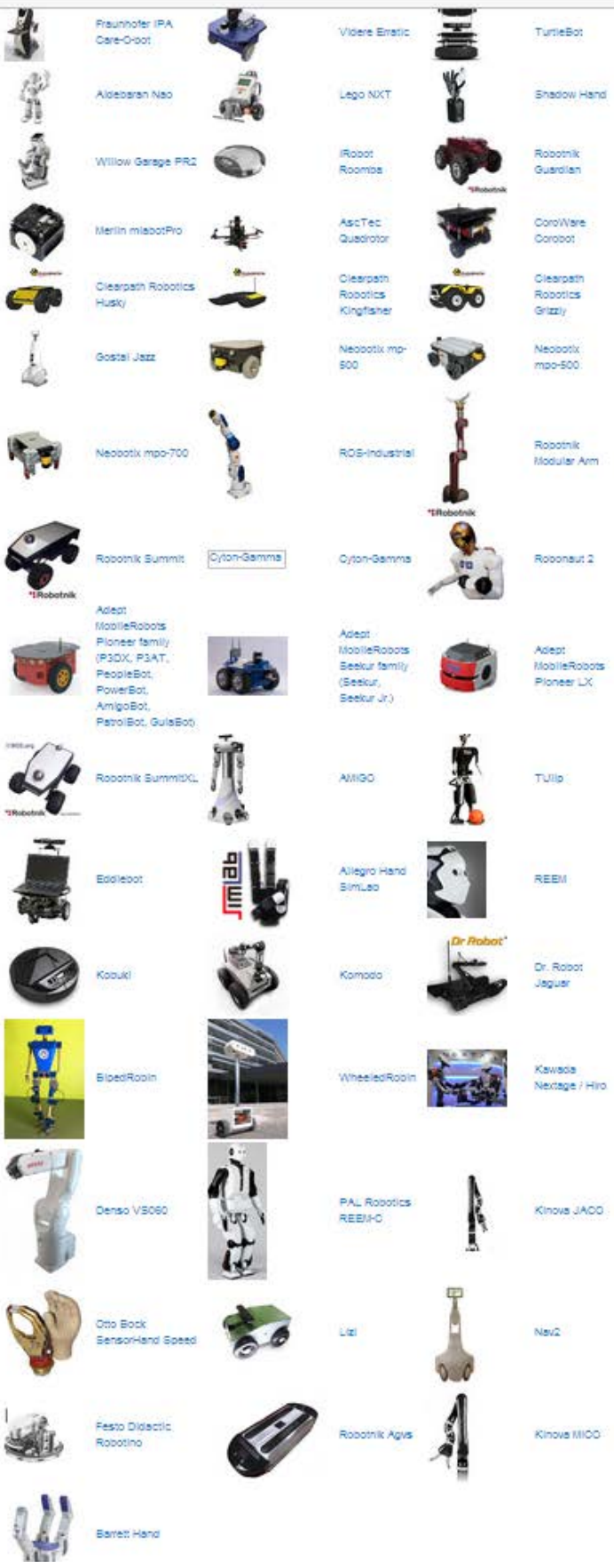



*Ilustración 47. Robots integrados en ROS.*

# Anexo 4. Instalación y configuración de ROS (PCL y Shadow Hand).

De acuerdo a lo expuesto en el capítulo 5, concretamente en el apartado *5.3.2. Instalación y Configuración del sistema.*, es posible encontrar gran cantidad de explicaciones y razonamientos para justificar el uso de ROS, partiendo de la comodidad ofertada por el middleware a la hora de embeber sistemas, y concluyendo con la activa comunidad de usuarios, que constituye el mayor servicio de soporte y ayuda posible.

Aprovechar esta última citación para insistir en la ayuda que puede encontrarse preguntando. Aunque, por lo general, no se suele estar acostumbrado a preguntar en un foro, ya sea por desconocimiento de esta filosofía, timidez, creer que no se va a ser escuchado… Este es el momento de señalar la profundidad de la equivocación que se comete con esta mentalidad, todos los desarrolladores tuvieron que aprender algún día, por ello son conocedores del esfuerzo que es preciso para ello y tratan de allanar el camino a los nuevos "caminantes" todo lo posible. Esta mentalidad y "estilo de vida" se encuentra ampliamente desarrollado y potenciado por las comunidades de cualquier software libre, encontrándose en los usuarios de ROS y PCL una gran prueba de ello. Tal y como señala Ortega y Gasset en su conocida frase:

*"Quien hace una pregunta teme parecer un ignorante durante cinco minutos. Quien no pregunta se mantiene ignorante toda la vida."*

*[José Ortega y Gasset]*

## 4.1. Instalación de ROS.

Antes de emprender el proceso de instalación es necesario haber escogido qué versión queremos instalar. A fecha de realización del presente documento, existen 5 versiones estables (Electro, Fuerte, Groovy, Hydro e Indigo (recién lanzada)) y una versión alfa (Jade). Esta decisión suele tomarse tras investigar durante unas horas las compatibilidades de los robots que deben ser embebidos y las funcionalidades requeridas. En el caso particular de este proyecto, se seleccionó groovy (aunque gran parte de los desarrollos se realizaron en Hydro por cuestiones de adaptabilidad y vistas al futuro) ya que el brazo robótico de Schunk con el que debía sincronizarse la mano únicamente se encontraba integrado en Groovy y Fuerte.

A continuación se detallará el proceso de instalación de Groovy, por medio del terminal de Ubuntu, como no podía ser menos. En caso de querer instalar Hydro bastaría con cambiar, en toda la nomenclatura, el vocablo "Groovy" por "Hydro". Así, sin más preámbulos, pasemos a abrir el terminal de Linux y teclear los siguientes comandos:

```
$    sudo sh -c 'echo ''deb http://packages.ros.org/ros/ubuntu precise main'' >
/etc/apt/sources.list.d/ros-latest.list'
```

Con esta directiva se incluyen los paquetes de ROS en la lista de Orígenes de Software. Sin entrar en detalles, existe una *"lista de sitios o repositorios"* a los que el sistema acude, cuando se solicita instalar un programa por terminal, para verificar si el paquete solicitado existe y puede ser adquirido. Nótese que este comando está incluyendo los paquetes para Ubuntu Precise (12.04 LTS).

A continuación, se notifica este cambio al sistema de gestión de paquetes y obtienen "las claves" de acceso al sitio de descarga.

```
$    wget https://raw.githubusercontent.com/ros/rosdistro/master/ros
.key -O - | sudo apt-key add -
```

El siguiente paso será actualizar la lista de repositorios del sistema (Ubuntu mantiene internamente una tabla con todos los paquetes que hay disponibles para instalar). Recordar que se han incluido los repositorios en el fichero de búsqueda, pero la copia en caché que mantiene el sistema para consulta rápida no ha sido modificada aún:

```
$    sudo apt-get update
```

Una vez configurada correctamente la procedencia de archivos, la siguiente acción a realizar es descargar e instalar ROS:

```
$    sudo apt-get install ros-groovy-desktop-full
```

Puesto que se desconocen, por falta de experiencia, todos los recursos que se necesitarán, se ha optado por adquirir la versión completa con todas las opciones y paquetes. En cierto punto de la instalación se nos preguntará si deseamos instalar ROS como un demonio del sistema (system's demon). Este término, que puede impresionar bastante, en Linux, es la forma mediante la que se define a los programas ejecutados en segundo plano y que forman parte de la lista de arranque del sistema operativo. Lo habitual es NO habilitar este modo, ya que provocaría un gasto de recursos permanente e innecesario.

Una vez instalado, se debe incluir la ruta de ROS en el archivo bashrc. Este archivo es el responsable de gran parte de los quebraderos de cabeza que surgirán durante el manejo de este middleware. Al igual que ha ocurrido con Ubuntu, ROS posee un listado de, entre otros parámetros, los paquetes que tiene disponibles para ejecutar así como de las librerías y su localización. De no incluirse la ruta en la que se han instalado los paquetes, al ser ejecutado, no podrá encontrarlos, lo que se traduce en incapacitación de arranque o correcto funcionamiento. Se podría entender que Linux posee las llamadas a ROS y éste, internamente, la localización de los paquetes a ejecutar en un fichero bashrc.

```
echo "source /opt/ros/hydro/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

Por último, se procesa una herramienta que, en ciertas ocasiones, será muy útil para adquirir y configurar nuevos paquetes y dependencias de ROS: *rosinstall*.

```
sudo apt-get install python-rosinstall
```

Ha llegado el momento de probar la instalación, para ello es necesario ejecutar el comando:

```
roscore
```

debiendo, por lo tanto, iniciarse ROS. Cuando se realice esta comprobación no debemos esperar que pase nada espectacular, simplemente aparecerá una lista de inicialización de procesos y el terminal quedará bloqueado ejecutando el núcleo de ROS, indicando que para la finalización del sistema es necesario ejecutar el comando:

```
rosout
```

Tal y como puede observarse en la siguiente imagen:

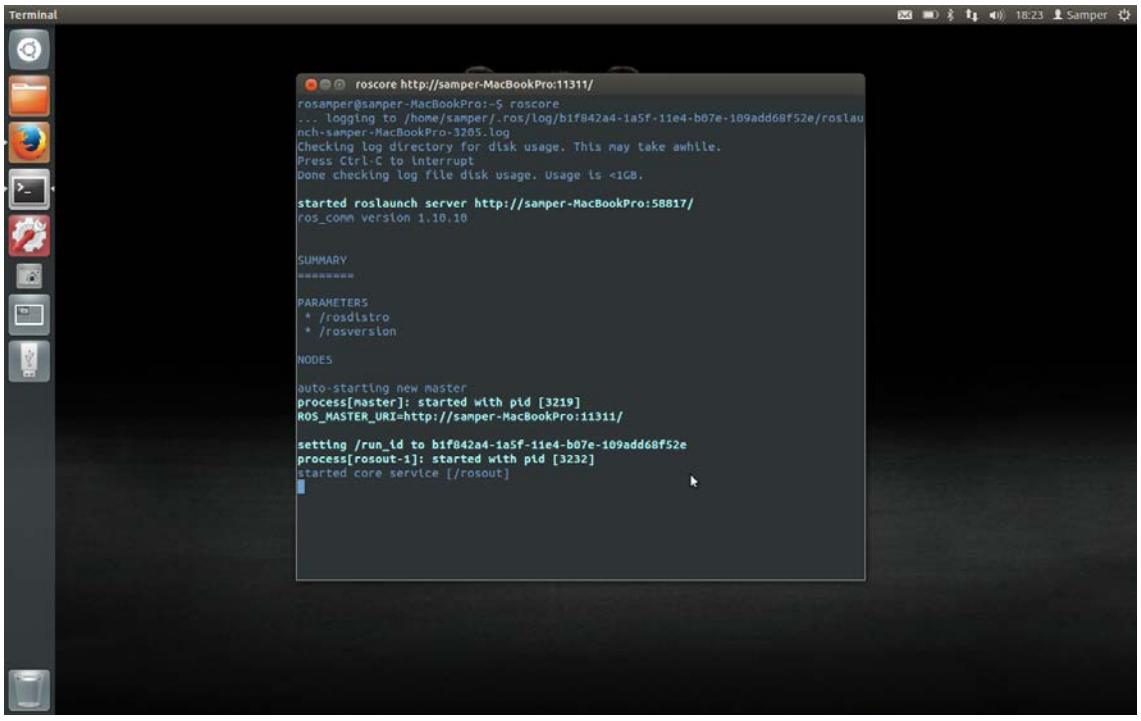



*Ilustración 48. Resultado Roscore*

Todo el proceso de instalación puede encontrarse y analizarse con mayor detalle en la página oficial [27]

27 HTTP://WIKI.ROS.ORG/HYDRO/INSTALLATION/UBUNTU

## 4.2. Instalación y configuración de los paquetes de Shadow.

Una vez asentado ROS en nuestro equipo, el siguiente paso natural a realizar será instalar los paquetes de Shadow para poder controlar la mano y usar el simulador gazebo. En Hydro, como se ha señalado en ciertas ocaciones, los desarrollos de ROS han avanzado más que en Groovy, en parte por las ventajas y correcciones de errores que éste ofrece respecto a su predecesor. Un motivo de esta migración, y evidente preferencia, puede encontrarse en las fechas de lanzamiento de ambas versiones, en torno a 4 meses de diferencia, Groovy fue lanzado con intención de introducir un nuevo paradigma o cambio de filosofía, Catkin. Por ello, presenta ciertos fallos, indicadores falta de adaptación. En cambio, Hydro trata de proporcionar una versión más estable de este nuevo paradigma (estrategia similar a la empleada con Windows 8 y Windows 8.1). Tanto es así, que, para Hydro, basta con ejecutar el siguiente comando para adquirir todos los paquetes y configuración necesaria que permite emplear las herramientas de Shadow:

```
sudo apt-get install ros-hydro-shadow-robot
```

En Groovy, por el contrario, es necesario realizar una serie de operaciones más complejas que involucran conocer los paquetes a instalar, entre otros aspectos. Dar las gracias a Toni Oliver, del equipo de Shadow, por su ayuda para la realización de este proceso.

En resumen, todos los pasos a seguir son:

```
$ sudo apt-get install ros-groovy-pr2-desktop ros-groovy-pr2-robot ros
-groovy-arm-navigation  ros-groovy-pr2-simulator ros-groovy-diagnostic
s-monitors python-rosinstall python-rosdep git -y


$ mkdir -p ~/ shadow_workspace
$ rosws init ~/ shadow_workspace /opt/ros/groovy
$ source ~/ shadow_workspace /setup.bash
$ cd ~/ shadow_workspace


$ rosws set  shadow_robot --git https://github.com/shadow-robot/sr-ros
-interface.git --v groovy-devel -y
```

```
$ rosws set  shadow_robot_ethercat --git https://github.com/shadow-rob
ot/sr-ros-interface-ethercat.git --v groovy-devel -y

$ rosws set  sr_visualization --git https://github.com/shadow-robot/sr
-visualization.git --v groovy-devel -y

$ rosws set  sr_config --git https://github.com/shadow-robot/sr-config
.git --v groovy-devel -y


$ rosws update


$ source ~/ shadow_workspace /setup.bash

$ echo source ~/ shadow_workspace /setup.bash >> ~/.bashrc


$ rosdep update -y


$ rosdep install shadow_robot -y

$ rosdep install shadow_robot_ethercat -y

$ rosdep install sr_visualization -y

$ rosdep install sr_config -y


$ rosmake shadow_robot

$ rosmake shadow_robot_ethercat

$ rosmake sr_visualization

$ rosmake sr_config


$ echo source `rosstack find sr_config`/bashrc/env_variables.bashrc >>
~/.bashrc
```

A continuación, se detallarán las acciones que se realizan a lo largo del script, con objeto de tener un mayor conocimiento sobre lo que está pasando en el sistema y favorecer el proceso de familiarización con el mismo. Señalar que este script así como el proceso de instalación para otras versiones se puede consultar en el apartado Shadow_robots de la página de ROS, actualizado recientemente. [28]

En primer lugar, se adquieren los paquetes en los que se basó la compañía Shadow para realizar el software de control:

28 HTTP://WIKI.ROS.ORG/SHADOW_ROBOT

```
$ sudo apt-get install ros-groovy-pr2-desktop ros-groovy-pr2-robot ros
-groovy-arm-navigation  ros-groovy-pr2-simulator ros-groovy-diagnostic
s-monitors python-rosinstall python-rosdep git -y
```

El siguiente paso es crear un directorio de trabajo de ROS en el que guardar todos estos paquetes que van a ser creados.

```
$ mkdir -p ~/shadow_workspace
$ rosws init ~/ shadow_workspace /opt/ros/groovy
```

No se debe olvidar incluir esta ruta en el archivo de fuentes de ROS:

```
$ source ~/ shadow_workspace /setup.bash
```

A continuación, se cambia el directorio de trabajo del Shell al que acaba de crearse y, seguidamente, se indican las direcciones de las que se desean descargar los paquetes (este tipo de descarga es conocida como *"svn" o "from source"*, el comando git se emplea ya que los directorios a descargar se encuentran en github.com). El comando *rosws set* es similar al empleado anteriormente para añadir dependencias y orígenes a Ubuntu, sólo que en este caso se adquiere todo el contenido de la url especificada de forma directa. Con el modificador set es posible dar un nombre al enlace.

```
$ rosws set  shadow_robot --git https://github.com/shadow-robot/sr-ros
-interface.git --v groovy-devel -y
$ rosws set  shadow_robot_ethercat --git https://github.com/shadow-rob
ot/sr-ros-interface-ethercat.git --v groovy-devel -y
$ rosws set  sr_visualization --git https://github.com/shadow-robot/sr
-visualization.git --v groovy-devel -y
$ rosws set  sr_config --git https://github.com/shadow-robot/sr-config
.git --v groovy-devel -y
```

Así, tras ejecutar estos comandos y proceder a la descarga, deberán aparecer cuatro carpetas en el espacio de trabajo creado (shadow_workspace) con la información que contienen los repositorios de cada una de estas url:

- Shadow_robot.

- Shadow_robot_ethercat.
- Sr_visualization.
- Sr_config.

Una vez establecidas las nuevas dependencias, se actualiza el fichero de posibles descargas:

```
rosws update
```

Por último, se actualiza y carga el fichero bashrc, para evitar problemas de actualización del mismo (de no hacerlo es posible que no encuentre algún paquete):

```
source ~/ shadow_workspace /setup.bash
$ echo source ~/ shadow_workspace /setup.bash >> ~/.bashrc
$ rosdep update -y
```

El siguiente paso consiste en descargar todos los paquetes modificados por shadow y prepararlos para su "instalación":

```
$ rosdep install shadow_robot -y
$ rosdep install shadow_robot_ethercat -y
$ rosdep install sr_visualization -y
$ rosdep install sr_config -y
```

A continuación, se procede a su construcción (equivalente a compilar):

```
$ rosmake shadow_robot
$ rosmake shadow_robot_ethercat
$ rosmake sr_visualization
$ rosmake sr_config
```

Y finalmente, se incorporan a la lista de paquetes en el sistema:

```
$ echo source `rosstack find sr_config`/bashrc/env_variables.bashrc >>
~/.bashrc
```

Llegados a este punto, es momento de comprobar si todo se ha instalado correctamente. Para ello, iniciamos el simulador, existen tres posibles opciones de ejecución, ya que estos paquetes contienen los modelos de dos robots: brazo y mano de Shadow (Shadow arm and Shadow hand):

- Simulador del brazo y la mano Shadow.

```
$ roslaunch sr_hand gazebo_arm_and_hand.launch
```

- Simulador de la mano Shadow Hand.

```
$ roslaunch sr_hand gazebo_hand.launch
```

- Simulador del brazo Shadow Arm.

```
$ roslaunch sr_hand gazebo_arm.launch
```

Se puede comprobar que los dos segundos son un despiece del primero, así si se ejecuta el primero, se iniciará Gazebo, emergiendo una pantalla de la forma:

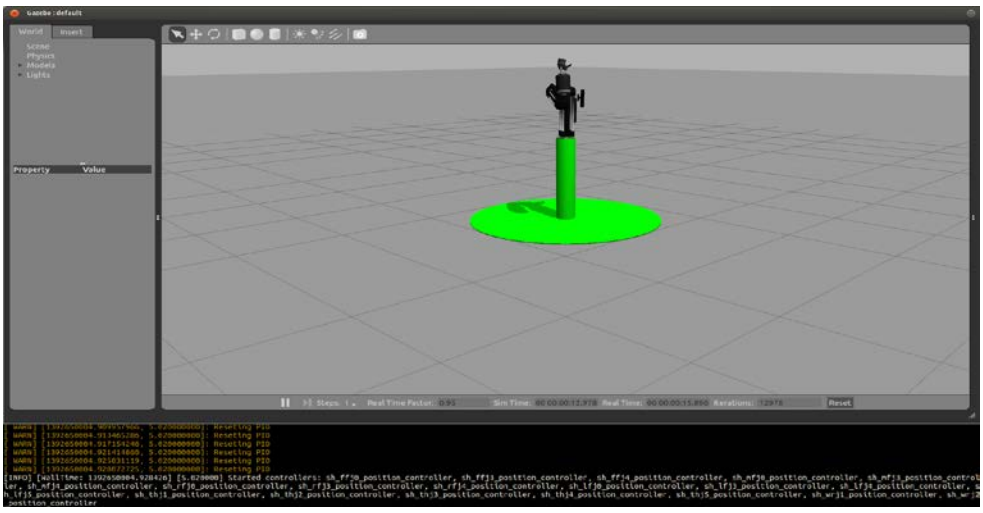


*Ilustración 49. Imagen Gazebo con modelo Shadow Hand and Arm*

En caso de dar algún error y no ejecutarse el simulador, conviene reiniciar Ubuntu y volver a probar. Si, simplemente, muestra un mensaje en el que se especifica un error de comunicación con el maestro, el procedimiento a seguir es finalizar la ejecución y volver a enviar el comando de puesta en marcha, esto es reiniciar gazebo.

A veces, la mano y/o el brazo aparecen caídos, como si no tuviesen controladores y no existiese fricción entre los componentes. Este error es debido a fallos relacionados con la tarjeta gráfica, para solventarlos basta con dirigirse al apartado de drivers privativos del menú de configuración y cambiar entre las distintas opciones de controladores gráficos que nos aparezcan.

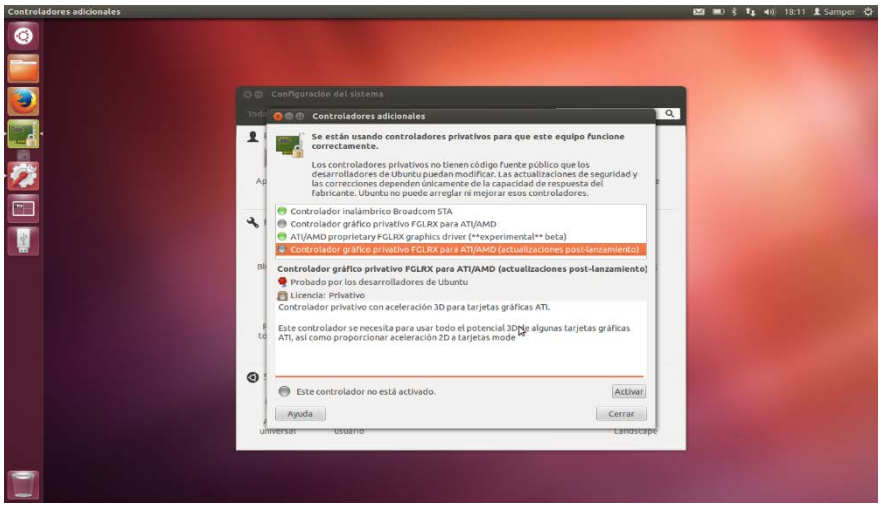



*Ilustración 50. Imagen de los controladores adicionales de video.*

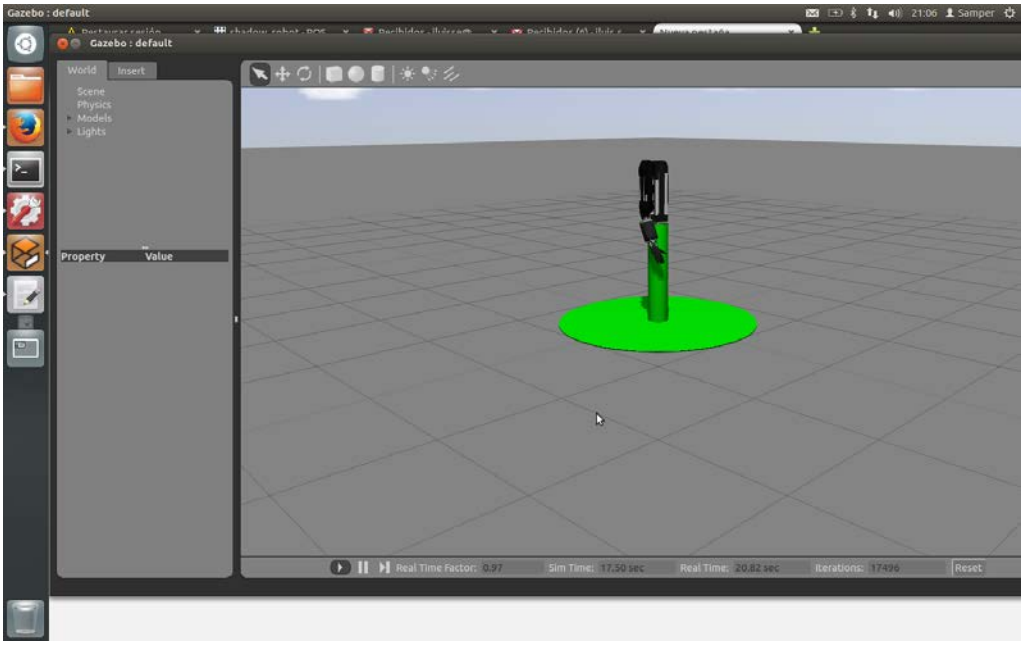



*Ilustración 51. Fallo de los controladores de Gazebo.*

En caso de producirse este error, tras estar usando correctamente el simulador. Probablemente se deba a alguna actualización realizada, la mejor forma de reiniciarlo es borrar todos los archivos de Shadow y repetir el proceso de instalación. A fecha de 24 de agosto de 2014 no es posible hacer uso del simulador ya que ha surgido algún problema con los modelos denotando problemas de estabilidad y fricciones (siguiente ilustración). Según fuentes de la empresa, este problema se espera que sea resuelto en poco tiempo.

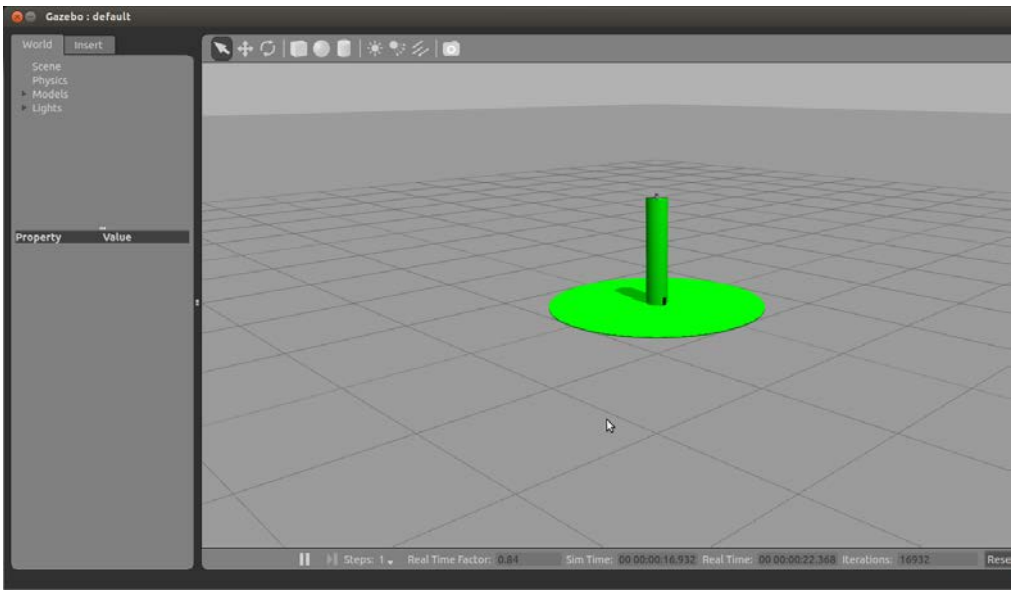



*Ilustración 52. Fallo de consistencia en el modelo.*

# Anexo 5. Primeros Pasos en ROS.

Una vez instaladas las herramientas básicas para comenzar con el desarrollo del proyecto, el siguiente movimiento a realizar, por imperativa obligación, es conocer el nuevo sistema que vamos a utilizar. Para ello, los propios desarrolladores de ROS, en su página, distribuyen una aglutinación de tutoriales con los que se pretende iniciar a los usuarios nóveles.

Se recomienda encarecidamente ojear, y si es posible, realizar cada uno de estos tutoriales que pueden encontrarse en la página oficial. [29] No obstante, a pesar de esta recomendación, este aparatado contempla un breve resumen de estos manuales a fin de refrescar conceptos para todos aquellos viejos conocidos del sistema que sólo buscan recordar ciertos aspectos esenciales.

Así mismo, durante dicho repaso, se han añadido algunas explicaciones y tratado de clarificar ciertos puntos que pueden resultar claves a la hora comprender el funcionamiento de este middleware. Todas estas definiciones "extra" son fruto de la experiencia propia del autor del TFG con ROS, caracterizada por "avanzar a ciegas" y marcada por el autoaprendizaje, quedando aún un largo camino por recorrer. En base a ello, no se pretende que sean tratadas como definiciones formales, sino que, por el contrario, sean consideradas como una ayuda o apoyo que buscan limar asperezas ya conocidas a futuros estudiantes.

## 5.1. Conceptos relativos al sistema de archivos.

Se puede decir que el elemento básico y fundamental de ROS es el paquete. Los paquetes pueden definirse como la unidad de organización de software de

29 http://wiki.ros.org/ROS/Tutorials

código ROS. Cada paquete puede contener bibliotecas, ejecutables, scripts u otros artefactos.

Inherentes a cada paquete se encuentran los archivos package.xml, también conocidos como manifiestos. Un manifiesto es una descripción de un paquete. En ellos se definen las dependencias entre paquetes y recopila meta-información sobre el paquete como la versión, desarrollador, soporte, mantenimiento, licencia, etc...

Si bien, a esta definición formal es posible añadir ciertas especificaciones extra. Todos el sistema ROS se encuentra formado por paquetes, en este aspecto, ROS puede concebirse como una base que podemos encontrar generalmente en la carpeta /opt/ros/-distro-. En esta carpeta se encuentran todos los paquetes que permiten a ROS funcionar, así como algunos que proporcionan funcionalidades añadidas. Como se pude observar en la imagen, estos paquetes se encuentran clasificados en carpetas. De entre todas ellas, generalmente se trabaja con la carpeta *stacks* (groovy), o *share* (hydro), en la que se suelen colocar los paquetes que ofrecen funcionalidades adicionales y han sido aceptados por la comunidad ROS. Excepcionalmente, en ciertas ocasiones, se acude a la carpeta *others* para buscar algún modelo cinemático o archivo singular, generalmente de gazebo, del

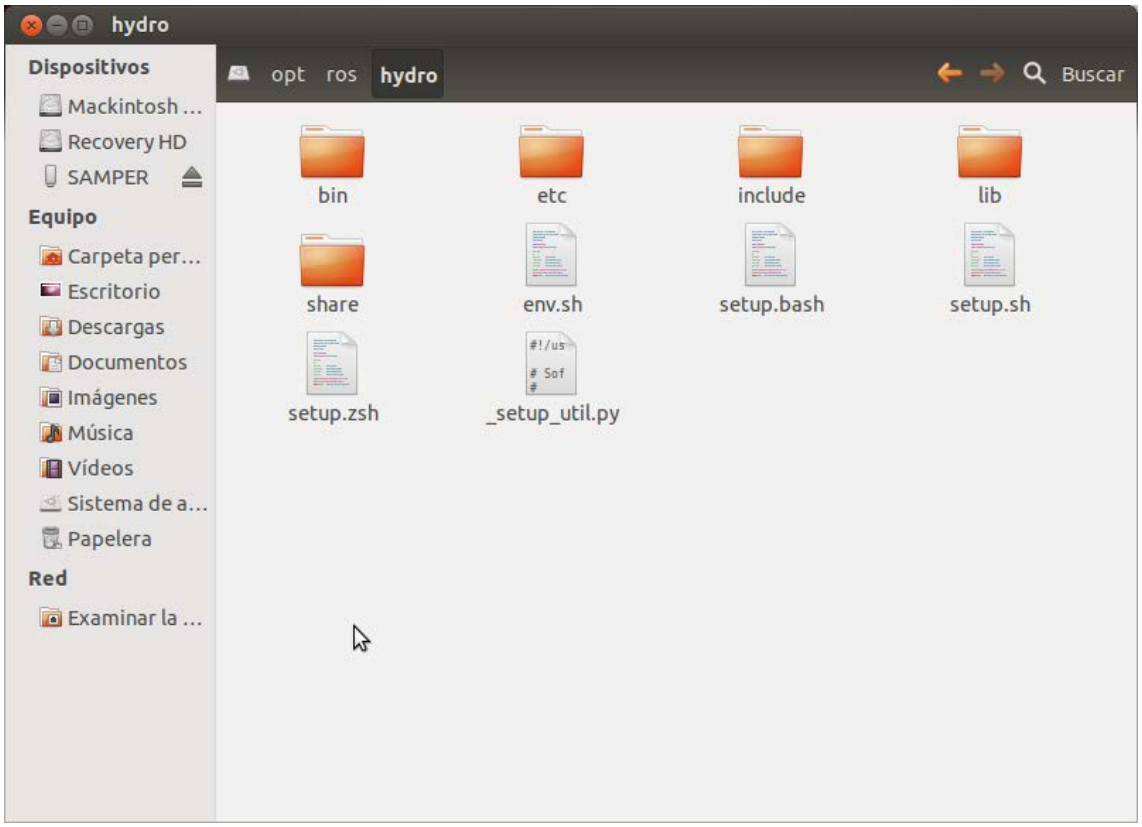



*Ilustración 53. Directorio principal ROS (/opt/ros/-distro- por defecto)*

que se precisa información, como podría ser el modelo de Kinect (Kinect.dae) del que se desea hacer un simulador.

Aunque esta es la carpeta de ROS, a la hora de trabajar, no guardaremos nuestros desarrollos en estas carpetas sino que definiremos un espacio de trabajo de ROS en nuestro directorio e indicaremos a ROS que busque ahí nuestro código, además de en las localizaciones habituales. De modo que, en resumidas cuentas, los paquetes pueden estar en dos localizaciones: archivos por defecto de ROS y espacios de trabajo del usuario.

Una vez fijada la localización de los mismos, conviene entender la jerarquía de los paquetes. De observar la imagen, se comprueba que dentro de cada paquete se tiene:

- Un descriptor del paquete. (package.xml).
- Una carpeta src en la que se encontrarán todos los archivos .py o .cpp a compilar.
- Un archivo CMakeLists.txt mediante el que se compilará el paquete de generación "automática".

- Una carpeta build, en la que se almacena toda la información necesaria para la compilación y uso del paquete.

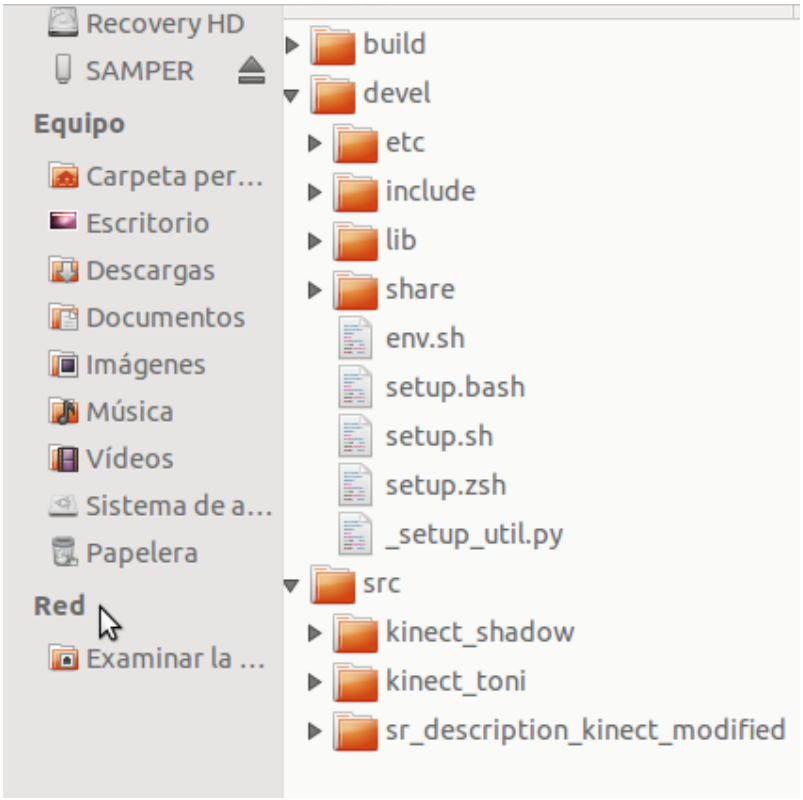



*Ilustración 54. Imagen de un workspace de ROS con tres paquetes de kinect desarrollados.*

Dadas estas características, se puede imaginar que el manejo de ROS implica la manipulación de paquetes. Puesto que gran parte de los problemas que encontraremos a la hora de crear uno de ellos, ejecutarlo o poner en funcionamiento cierta aplicación estarán relacionados con la presencia, o ausencia, de paquetes, el propio sistema consta de una lista de comandos que permiten detectar fallos o anomalías sin necesidad de navegar por las infinitas carpetas que lo constituyen. Entre ellos, destacan, por su extrema utilidad, los siguientes:

- ***rospack*** permite obtener información acerca de los paquetes. Para obtener, por ejemplo, la ruta de acceso al paquete.

Uso:

```
# Rospack find [nombre_paquete]
```

Ejemplo:

```
$ Rospack find roscpp
```

Lo que devolvería:

- `Nuestro_ruta_a_ros / share / roscpp`

Si hemos instalado hydro de la forma que se especificó:

- `/ Opt / ros / hydro / share / roscpp`

- ***roscd*** es parte de la suite *Rosbash*. Este comando nos permite cambiar directamente al directorio de un paquete o una pila. Puede definirse como la integración del comando cd de Linux en ROS. La ventaja respecto al comando cd habitual es que permite trasladarse a la localización de un cierto paquete, sin necesidad de conocer la ruta.

Uso:

```
# Roscd [localizacion [/ subdir]]
```

Ejemplo práctico:

```
$ Roscd roscpp
```

De este modo, el directorio de trabajo del Shell en el que se esté trabajando se traslada a la localización de este paquete.

Se debe tener en cuenta que roscd, al igual que otras herramientas de ROS, sólo será capaz de encontrar paquetes de ROS que estén dentro de los directorios listados en su ROS_PACKAGE_PATH. Para ver lo que se encuentra en su ROS_PACKAGE_PATH:

```
$ Echo $ ROS_PACKAGE_PATH
```

- ***rosls*** es parte de la Rosbash suite. Al igual que ocurría con el caso anterior, se corresponde con un comando equivalente en Linux: Ls.

En este caso, permite acceder directamente al contenido de un paquete por su nombre en lugar de tener que emplear vía de acceso absoluta, o relativa.

Uso:

```
# rosls [Localización [/ subdir]]
```

Ejemplo:

```
$ Rosls roscpp_tutorials
```

Resultado de retorno:

```
• cmake package.xml srv
```

## 5.2. Creación de nuestro espacio de trabajo.

A la hora de organizar el entorno de programación, podemos encauzar nuestras andadas por dos caminos:

- Modificar el sistema directamente, añadiendo nuestros paquetes como dependencias quedando así protegidos por privilegios especiales de administración.
- Crear un espacio de trabajo en nuestro sistema de modo que tengamos, en un lugar aislado y debidamente separado, el núcleo de ROS y, en carpetas manejables sin privilegios especiales, el código desarrollado por nosotros o terceros.

Aunque las diferencias son claras, es posible añadir algunas más. Con el primer procedimiento se evitan gran parte de los problemas de dependencias y linkado que aparecerán conforme avancemos en el manejo de ROS y que darán lugar a una inigualable e inimaginable cantidad de quebraderos de cabeza. Por otro lado, en caso de querer borrar algún código o paquete erróneo deberemos lidiar más con el sistema. En cuanto al segundo enfoque, puede resumirse en dos frases conocidas por todos aquellos que hayamos cursado asignaturas de programación concurrente y tiempo real: “divide y vencerás” y “hay que ser ordenado”.

En este caso, y en el de la gran mayoría de programadores, a excepción de las grandes mentes de este campo, se opta por adoptar una postura menos entusiasta y más ordenada, aunque ello implique más trabajo a la hora de crear paquetes.

Para crear el espacio de trabajo de *catkin* basta con dirigirnos a la ruta en la se desee tenerlo y crear las carpetas oportunas, a continuación, se declara esta carpeta como espacio de trabajo de ros de tipo *Catkin*:

```
$ Mkdir-p ~ / nombre_del_workspace / src
$ Cd ~ / nombre_del_workspace / src
$ Catkin_init_workspace
```

A pesar de que el espacio de trabajo está “vacío”, puesto que existe un archivo CMakeLists.txt, es posible construirlo de modo que se generen todos los archivos de configuración de este “paquete especial”:

```
$ Cd ~ / nombre_del_workspace /
$ Catkin_make
```

El comando *catkin_make* es la herramienta de trabajo esencial para trabajar con espacios de trabajo de tipo *catkin*. Si nos fijamos en el directorio actual, debería tener ahora un subdirectorio 'build' y una carpeta 'devel'. Asimismo, dentro de la carpeta 'devel' se pueden encontrar varios archivos setup.* de distinta extensión. Por último, se debe incluir esta ruta en los “lugares” en los que ROS busca sus paquetes, para ello:

```
$ Source devel / setup.bash
```

En relación a este tema, se debe señalar que existe un archivo denominado *~/.bashrc* en el que se encuentra la lista de posibles localizaciones de un paquete. Si se abre, con cualquier editor de texto, se puede comprobar como al final del mismo se sitúan, en caso de sólo haber realizado este tutorial, las siguientes líneas o, en su defecto, algunas de forma similar o equivalente:

```
$   Source /opt/ros/distro/setup.bash
$   Source directorio de Shadow (~/workspace_ws por defecto, en
nuestro  caso shadow_workspace)
$   Source devel / setup.bash
```

Finalmente nuestro directorio de trabajo tendrá una estructura de la forma:

```
workspace_folder / - ESPACIO DE TRABAJO
  src / - ESPACIO FUENTE
    CMakeLists.txt - archivo CMake "Nivel Superior", proporcionado
por el método catkin empleado para su creación
    package_1 /
      Archivo CMakeLists.txt para package_1 - CMakeLists.txt
```

```
    package.xml - Paquete manifiesto para package_1
  ...
  package_n /
    Archivo CMakeLists.txt para package_n - CMakeLists.txt
    package.xml - Paquete manifiesto para package_n
```

## 5.3. Creación de paquetes.

Aunque en el punto anterior se han señalado varios elementos como constituyentes básicos de un paquete, se debe notar que no todos ellos son necesarios, siendo tan sólo indispensables su descriptor (package.xml) y el archivo con las directivas de preprocesador y de compilación *CMakeLists.txt.*

Hasta el momento se ha hablado de *catkin* como forma de compilar y crear paquetes. No obstante, éste no ha sido introducido ni comparado con otras opciones de gestión y creación de paquetes. A la hora de crear un paquete se distingue entre dos procedimientos: *rosbuild*, heredado de las primeras versiones y mantenido por cortesía (y dada la amplia demanda) y *catkin*, nuevo cambio en el paradigma de programación introducido con la versión *Groovy.* En este caso concreto, se abordarán todos los aspectos relativos a la configuración, creación y uso de paquetes desde la filosofía de *catkin,* ya que se impondrá de forma íntegra con las próximas versiones que darán lugar a la desaparición de *rosbuild.* Dicho lo cual y siguiendo el planteamiento lógico habitual, lo primero que se necesita de un paquete para que este pueda ser usado es su existencia como tal, para ello seguimos los siguientes pasos:

Accedemos a la carpeta *src* de nuestro directorio de trabajo habitual (nuestra área de trabajo de ROS o workspace):

```
$ Cd ~ / catkin_ws / src
```

A continuación, se emplea la herramienta para creación de paquetes que proporciona *catkin*:

```
# Catkin_create_pkg <nombre_de_paquete> [depend1] [depend2] [depend3]
```

Un ejemplo de aplicación sería:

```
$ Catkin_create_pkg mi_primer_paquete std_msgs Rospy roscpp
```

Este comando creará un paquete de nombre "mi_primer_paquete" en el directorio citado, los siguientes tres nombres corresponden a las dependencias del mismo paquete. En este caso de ejemplo, podemos ver que el paquete creado depende de la librería de Python, la de C++ y una para el envío de mensajes, por lo que este paquete corresponderá a una aplicación que involucre manejo de nodos básicos. Al crear un paquete, se genera un *CMakeLists.txt* automáticamente, si conocemos las dependencias que tendrá nuestro paquete podemos incluirlas durante su generación evitando tener que modificar este archivo posteriormente.

Las dependencias suelen ser también una fuente de problemas ya que de no incluirlas durante la creación del paquete, o cometer un error al adherirlas, es necesario modificar el archivo *CMakeLists.txt* generado. Siendo en muchos casos, más rentable crear un nuevo paquete y migrar los archivos fuente, localizados en *src*. Para lidiar con este tipo de altercados, ROS permite conocer todas las dependencias que se añadieron a un paquete mediante la herramienta ya presentada en puntos anteriores Rospack:

```
$ Rospack depends1 mi_primer_paquete
```

```
std_msgs
Rospy
roscpp
```

Con el modificador *depends1*, se obtienen únicamente aquellas dependencias que se agregaron durante la creación del paquete, mediante el comando usado por el usuario. Si, por el contrario deseamos conocer todas las dependencias existentes:

```
$ Rospack depends mi_primer_paquete
cpp_common
rostime
roscpp_traits
roscpp_serialization
genmsg
genpy
message_runtime
rosconsole
std_msgs
rosgraph_msgs
xmlrpcpp
roscpp
rosgraph
catkin
rospack
roslib
Rospy
```

La explicación de la aparición espontánea de un amplio conjunto de dependencias, en un paquete en cuya creación se incluyeron sólo tres y que no ha sido modificado posteriormente, se encuentra en las inclusiones por defecto que realiza el propio comando al crear el paquete, ya que constituyen dependencias imperativas para el correcto manejo del paquete por el sistema.

En el tutorial correspondiente de la página oficial, se puede encontrar más información relativa a la creación de paquetes y modificación del fichero *CMakeLists.txt* que permitan lograr una mayor personalización. [30]

[30] http://wiki.ros.org/ROS/Tutorials/CreatingPackage

## 5.4. Construcción y manejo del paquete.

Una vez creado el paquete, incluiremos nuestros archivos .cpp en la carpeta src del paquete. Tras haber hecho esto, tan sólo queda construirlo.

En nuestro caso, la experiencia con la programación hasta este momento se había basado en el uso de entornos de desarrollo principalmente, los cuales reducen la ardua tarea de compilar a pulsar un botón. No obstante, durante la fase de manejo y control del robot Shadow Hand se adquirieron competencias relacionadas con el uso de archivos *MakeFile* para compilación, así como experiencia con la compilación por comandos. Se recomienda pues, leer algún manual de creación de archivos de compilación en Linux, o, en su defecto, consultar las secciones en las que se detalla el uso de la Shadow Hand con API's de control, a fin de adquirir cierto conocimiento en cuanto MakeFiles y Cmakes se refiere, antes de proseguir con la lectura del presente anexo. En el anexo 2 se puede encontrar información sobre creación de MakeFiles.

Para la compilación, ROS proporciona un comando muy sencillo a la par que poderoso denominado *catkin_make*, este comando realiza las siguientes operaciones, al ser ejecutado en un directorio que contenga un archivo CMakeLists.txt:

```
# In a CMake project
$ mkdir build
$ cd build
$ cmake ..
$ make
$ make install  # (optionally)
```

Por lo que, para construir el paquete, bastaría con dirigirse al espacio de trabajo de *catkin* y teclear:

```
# In a catkin workspace
$ catkin_make --source my_src
$ catkin_make install --source my_src  # (optionally)
```

Este es uno de los inconvenientes de *catkin*: cada vez que compilamos un paquete, en nuestro espacio de trabajo, se revisan todos los paquetes presentes en dicha área, lo que a veces puede ocasionar la compilación de un programa a medio finalizar o con problemas de dependencias. Este hecho queda traducido en una lista de errores en el terminal que a voz de pronto no harán sino sobresaltar al programador.

Si todo ha ido bien, al compilar el espacio de trabajo que acabamos de crear, obtendremos un listado de información similar al siguiente:

```
Base path: /home/user/mi_espacio_de_trabajo
Source space: /home/user/ mi_espacio_de_trabajo /src
Build space: /home/user/ mi_espacio_de_trabajo /build
Devel space: /home/user/ mi_espacio_de_trabajo /devel
Install space: /home/user/ mi_espacio_de_trabajo /install
####
#### Running command: "cmake /home/user/ mi_espacio_de_trabajo /src
-DCATKIN_DEVEL_PREFIX=/home/user/ mi_espacio_de_trabajo /devel
-DCMAKE_INSTALL_PREFIX=/home/user/ mi_espacio_de_trabajo /install"
in "/home/user/catkin_ws/build"
####
-- The C compiler identification is GNU 4.2.1
-- The CXX compiler identification is Clang 4.0.0
-- Checking whether C compiler has -isysroot
-- Checking whether C compiler has -isysroot - yes
-- Checking whether C compiler supports OSX deployment target flag
-- Checking whether C compiler supports OSX deployment target flag
- yes
-- Check for working C compiler: /usr/bin/gcc
-- Check for working C compiler: /usr/bin/gcc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Using CATKIN_DEVEL_PREFIX: /tmp/ mi_espacio_de_trabajo /devel
-- Using CMAKE_PREFIX_PATH: /opt/ros/hydro
-- This workspace overlays: /opt/ros/hydro
```

```
-- Found PythonInterp: /usr/bin/python (found version "2.7.1")
-- Found PY_em: /usr/lib/python2.7/dist-packages/em.pyc
-- Found gtest: gtests will be built
-- catkin 0.5.51
-- BUILD_SHARED_LIBS is on
-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
-- ~~  traversing packages in topological order:
-- ~~  - beginner_tutorials
-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
-- +++ add_subdirectory(beginner_tutorials)
-- Configuring done
-- Generating done
-- Build files have been written to: /home/user/ mi_espacio_de_trab
ajo /build
####
#### Running command: "make -j4" in "/home/user/ mi_espacio_de_trab
ajo /build"
####
```

Vemos que, en primer lugar, el sistema carga una serie de comandos y realiza ciertas tareas de precompilación. Posteriormente, éste realiza una lista de los paquetes a compilar y finalmente arroja los resultados obtenidos en la compilación de cada paquete. El problema se haya en que de encontrarse un error al compilar un paquete, se detendrá la ejecución de la orden y todos aquellos paquetes que tengan un orden inferior en la lista no serán compilados hasta que se solventen los problemas que han producido el error.

## 5.5. Ros y los nodos.

Para las explicaciones futuras de este capítulo se emplea un simulador creado por los desarrolladores de ROS, en un principio, para controlar el primer robot que funcionó bajo este sistema. Como tradición, guiño al pasado o memoria de sus primeros pasos, este equipo de desarrolladores ha adaptado este diseño para iniciar a todos los programadores nóveles del sistema.

Para instalar el simulador debemos escribir desde el terminal, cambiando distro por la versión de ROS en la que nos encontramos, el siguiente comando:

```
$ Sudo apt-get install Ros <distro> -ros-tutorials
```

Una vez preparado el sistema, pasamos a explicar uno de los conceptos más comentados al hablar de ROS: los nodos. Si bien "físicamente" el sistema se constituye por paquetes, al ser ejecutado, éste desarrolla un entramado de nodos que se comunican entre sí como si de una red neuronal o red cristalina de vibración se tratase.

Un nodo, en realidad, no es mucho más que un archivo ejecutable dentro de un paquete de ROS. Los nodos de ROS utilizan una biblioteca *cliente de ROS* para comunicarse con otros nodos. Además, están dotados de capacidad para publicar o suscribirse a un tema (topic). Por último, los nodos también pueden proporcionar o utilizar un servicio. Para realizar dicha comunicación, los nodos de ROS emplean librerías de Python y C++, lo que reduce el número de posibilidades de programar en ROS a dos.

El primer nodo que debe ejecutarse al realizar cualquier tarea en ROS es el nodo maestro.

```
$ Roscore
```

Generándose un texto informativo similar al mostrado a continuación:

```
... logging to ~/.ros/log/9cf88ce4-b14d-11df-8a75-00251148e8cf/rosl
aunch-machine_name-13039.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://machine_name:33919/
ros_comm version 1.4.7

SUMMARY
========

PARAMETERS
 * /rosversion
 * /rosdistro

NODES

auto-starting new master
process[master]: started with pid [13054]
ROS_MASTER_URI=http://machine_name:11311/

setting /run_id to 9cf88ce4-b14d-11df-8a75-00251148e8cf
process[rosout-1]: started with pid [13067]
started core service [/rosout]
```

Es fácil imaginar que una vez iniciado este comando, a medida que se inicien nuevos "programas", la población de nodos activos en nuestro sistema proliferará de forma vírica. Puesto que conviene tener el crecimiento de cualquier tipo de población controlado, ROS proporciona un comando para ello:

```
$ rosnode list
```

En este caso sólo se tiene un nodo en ejecución, por lo que mostrará:

```
/rosout
```

En otras ocasiones, es necesario contar con más información sobre la actividad de un nodo, para lo que se recurre al comando *rosnode info*:

```
$ rosnode info /rosout
```

```
--------------------------------------------------------------------------------
Node [/rosout]
Publications:
 * /rosout_agg [rosgraph_msgs/Log]

Subscriptions:
 * /rosout [unknown type]

Services:
 * /rosout/set_logger_level
 * /rosout/get_loggers

contacting node http://machine_name:54614/ ...
Pid: 5092
```

En este momento ya es posible ejecutar cualquier paquete de ROS. No se debe olvidar nunca tener un maestro activo, ya que éste gobierna la creación de nodos y se encarga de establecer las vías de comunicación que deben utilizar los mismos, así como los temas a los que se encuentran asociados, ya sea como publicadores o subscriptores. Sin más preámbulos, para lanzar un nodo de un cierto paquete se emplea el comando *rosrun*:

```
$ Rosrun [nombre_paquete] [nombre_nodo]
```

De esta forma, si se quiere ejecutar el nodo *turtlesim_node*, correspondiente al paquete turtlesim, debe emplearse la siguiente orden:

```
$ Rosrun turtlesim turtlesim_node
```

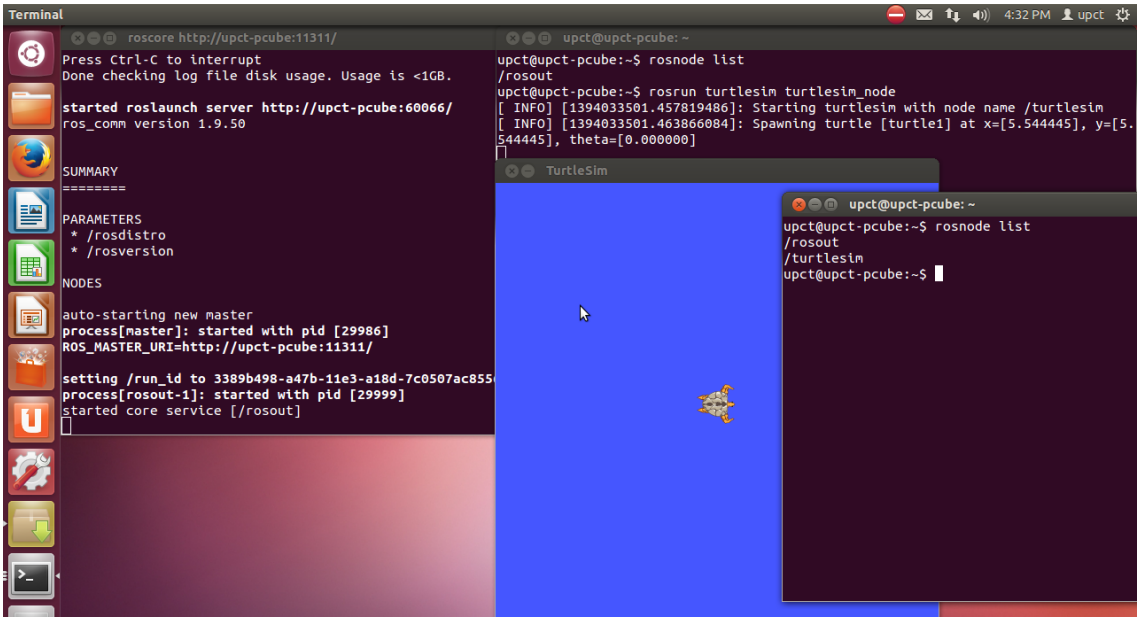



*Ilustración 55. Llamada a turtlesim_bot*

Al ejecutar este comando en una celda de terminal, emergerá una ventana con una tortuga. Como curiosidad, denotar que la tortuga es diferente cada vez que se llama a este comando ya que se genera de forma aleatoria. Esta tortuga, aparentemente sencilla, es el simulador de un robot real denominado turtlebot y desarrollado por Willow Garage. Aunque gráficamente destaca por su sencillez, a nivel de programación de trayectorias y potencial móvil sorprende por su complejidad.

En la imagen se puede apreciar como la información de nodos activos ha variado respecto a la situación inicial.

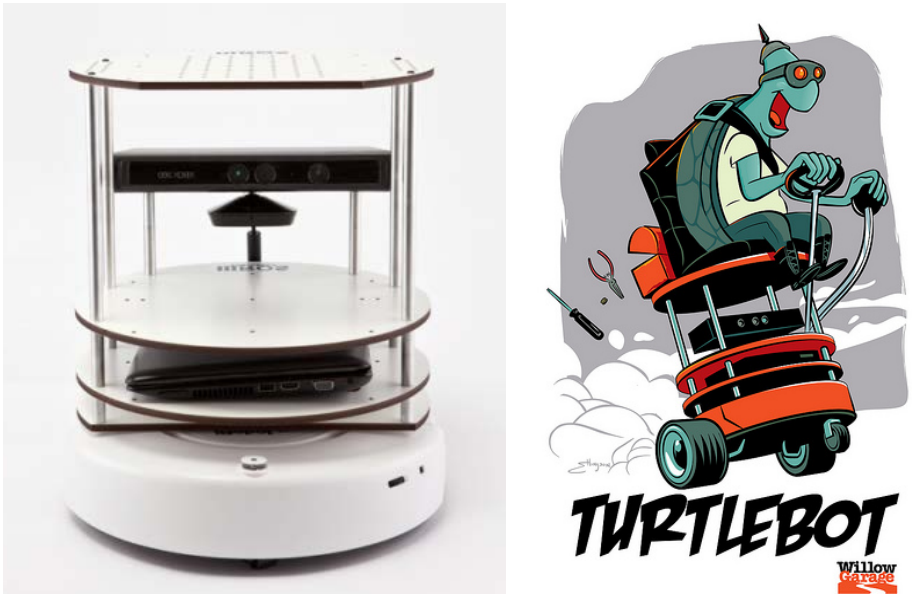



*Ilustración 56. Turtlebot de Willow Garage*

Otra opción de interés, que favorece el reconocimiento de los nodos en ejecución, es proporcionales un nombre establecido por el usuario en lugar de la nomenclatura establecida por defecto, ya sea por el sistema o el desarrollador del paquete. De este modo, entre la amplia variedad de modificadores que presenta rosrun, encontramos *__name:=*

Ejemplo de uso:

```
$ Rosrun turtlesim turtlesim_node __name: = my_turtle
```

Si se vuelven a listar los nodos activos:

```
$ rosnode list
```

Se podrá ver algo similar a:

```
/ Rosout
/ My_turtle
```

Para concluir, señalar que; para aquellas situaciones en las que es necesario conocer de antemano si el nodo está activo y disponible para usar, se presenta el comando *ping*:

```
$ Ping rosnode my_turtle
```

```
rosnode: node is [/my_turtle]
pinging /my_turtle with a timeout of 3.0s
xmlrpc reply from http://aqy:42235/     time=1.152992ms
xmlrpc reply from http://aqy:42235/     time=1.120090ms
xmlrpc reply from http://aqy:42235/     time=1.700878ms
xmlrpc reply from http://aqy:42235/     time=1.127958ms
```

## 5.6. Introducción al uso de topics.

Para este apartado es necesario haber instalado el paquete rqt_graph:

```
$ sudo apt-get install ros-<distro>-rqt
$ sudo apt-get install ros-<distro>-rqt-common-plugins
```

Una vez hecho esto, lanzamos los nodos necesarios:

```
$ roscore
$ rosrun turtlesim turtlesim_node
$ rosrun turtlesim turtle_teleop_key
```

Este último comando permite controlar el movimiento de la tortuga mediante las flechas del teclado, tras activarse el terminal quedará con la siguiente información

```
[ INFO] 1254264546.878445000: Started node [/teleop_turtle], pid [5
528], bound on [aqy], xmlrpc port [43918], tcpros port [55936], log
ging to [~/ros/ros/log/teleop_turtle_5528.log], using [real] time
Reading from keyboard
---------------------------
Use arrow keys to move the turtle.
```

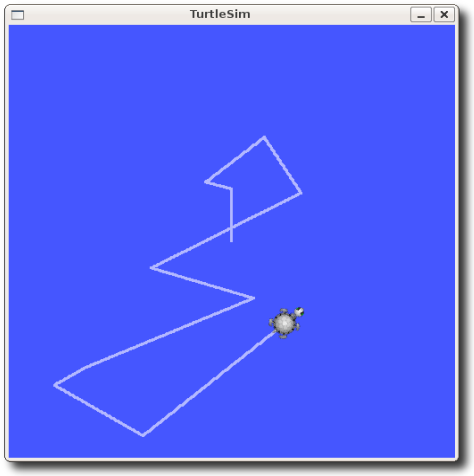



*Ilustración 57. Movimiento de Turtlebot con asistente para teclado.*

Aunque parece un script tradicional de control, en realidad, el nodo turtlesim (la tortuga) se está comunicando con el nodo de tele-operación mediante un topic de ROS. Concretamente, el panel de tele-operación publica información en el topic y la tortuga la puede leer al estar subscrita. Es importante saber que esta relación publicador-subscriptor, así como el tema de relación son creados por el nodo maestro roscore.

La herramienta *rqt_graph,* instalada anteriormente, permite obtener información gráfica que aclare estos nuevos conceptos de comunicación presentados, entre otros aspectos. Para su ejecución es necesario emplear:

```
$ rosrun rqt_graph rqt_graph
```

Al ejecutar esta sentencia, se abrirá una nueva ventana con toda la información de los topics abiertos, incluyendo los nodos adscritos, presentada de forma gráfica:

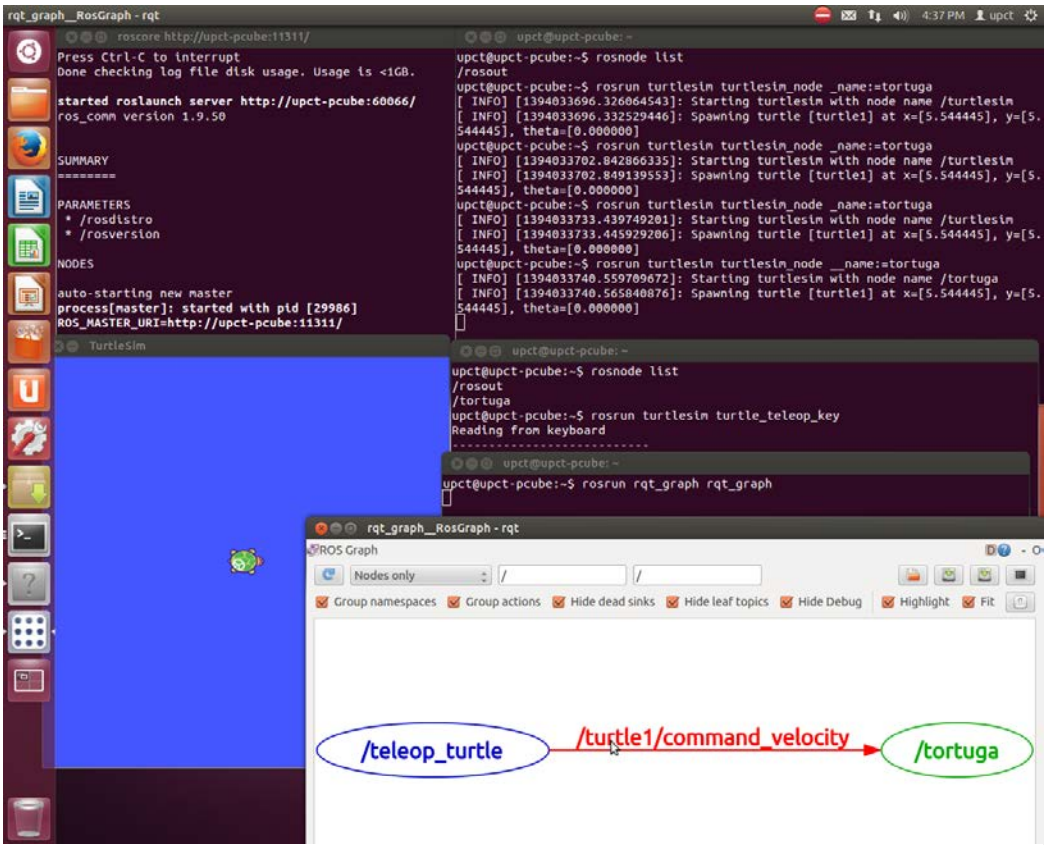



*Ilustración 58. Ejemplo de uso de rqt_graph*

Vemos que, en este caso, sólo se encuentra en ejecución, sin tener en cuenta al maestro, un nodo teleop mandando comandos de velocidad a nuestra tortuga.

Otro comando de gran valía al tratar con topics es *rostopic echo*. Esta orden permite consultar "qué está pasando en el topic" en tiempo real:

```
$ rostopic echo /turtle1/cmd_vel
```

Para groovy sería*:*

```
$ rostopic echo /turtle1/command_velocity
```

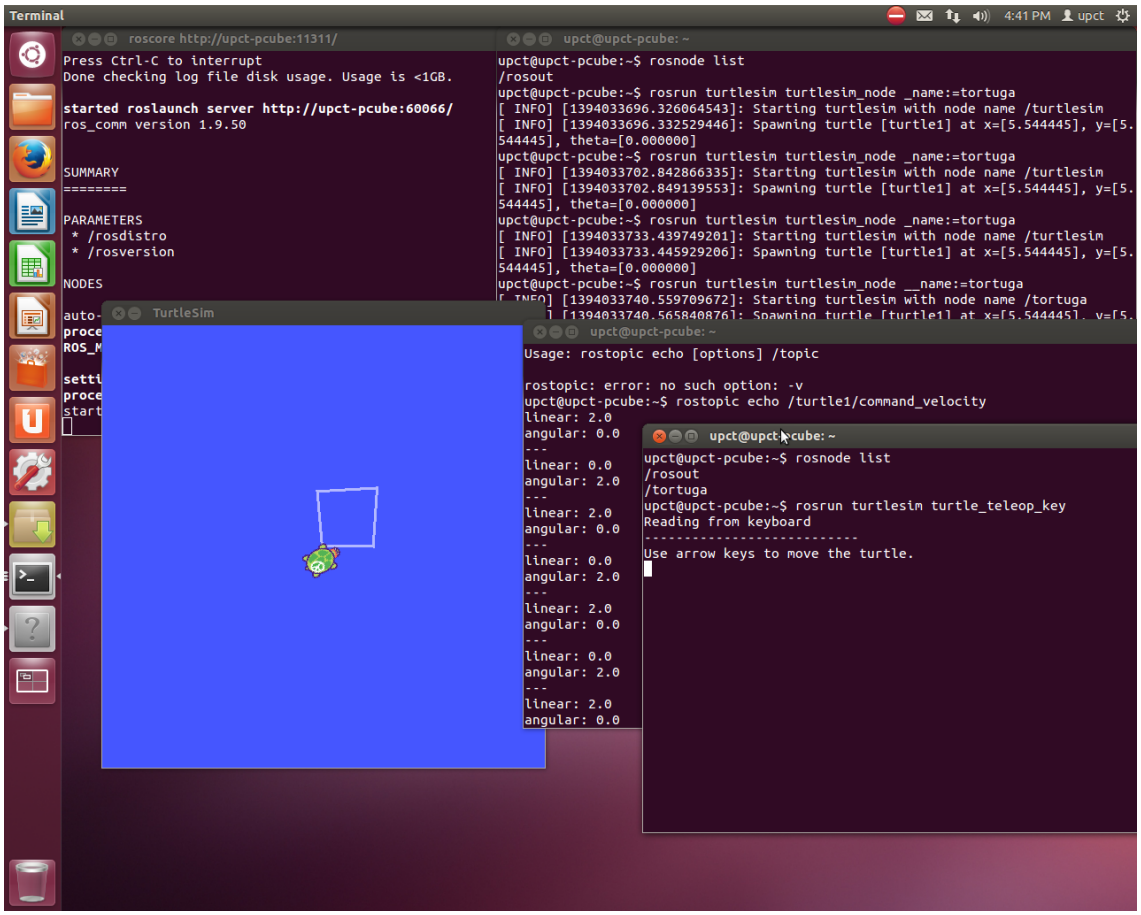



*Ilustración 59. Uso de rostopic echo.*

Se comprueba que, al escanear el topic, aparecen todas las posiciones sucesivas que ha ido alcanzando la tortuga. Si se probase, en este momento, el comando rqt_graph visto anteriormente, se podría verificar que el número de relaciones ha aumentado.

Al igual que ocurría con los nodos, para los topics también existe un comando que permite escanearlos:

```
$ rostopic list
```

De su empleo en modo *verbose* se obtienen, adicionalmente, de los topics existentes, los publicadores y subscriptores:

```
$ rostopic list -v
```

```
Published topics:
 * /turtle1/color_sensor [turtlesim/Color] 1 publisher
 * /turtle1/command_velocity [turtlesim/Velocity] 1 publisher
 * /rosout [roslib/Log] 2 publishers
 * /rosout_agg [roslib/Log] 1 publisher
 * /turtle1/pose [turtlesim/Pose] 1 publisher

Subscribed topics:
 * /turtle1/command_velocity [turtlesim/Velocity] 1 subscriber
 * /rosout [roslib/Log] 1 subscriber
```

Ya han sido presentados los topics y señalado que mediante ellos se establece la comunicación entre los distintos nodos del sistema, puediendo éstos actuar como subscriptores y publicadores como si de un foro se tratase. Estas publicaciones y subscripciones no son sino mensajes enviados desde un nodo a otro. Además, estos mensajes, pueden ser de disinto tipo; desde notificaciones sin contenido empleadas como avisos ( como ocurre cuando se realizan implementaciones o integraciones en ROS y se necesitan balizas de sincronización y latencias) a cadenas de caracteres, pasando por números de tipo float, parámetros enteros …

Se llega así a la primera limitación o imposición a la hora de crear topics y su respectivos nodos, el topic debe ser del mismo tipo que los mensajes publicados en él. O dicho de otra forma, en cada topic sólo se pueden intercambiar mensajes de un mismo tipo lo que, a su vez, determinará el tipo de nodos adheridos, limitando las opciones de creación.

Para saber el tipo de mensajes que acepta un cierto topic, se hace uso del comando *rostopic* y del modificador *type*, de la forma:

```
$ rostopic type /turtle1/cmd_vel
```

Devolviendo el tipo de mensajes que se debe emplear en las comunicaciones con los nodos adscritos al topic de velocidad de la tortuga, en este caso:

```
geometry_msgs/Twist
```

Ha llegado el momento de mover la tortuga mediante líneas de comandos, esto es, publicaciones directas. A la hora de publicar en un cierto topic se contemplan dos opciones:

- Publicación instantánea y ocasional mediante línea de comandos.
- Crear un publicador propio (programando).

De momento, se comenzará por la opción más sencilla. A veces, como puede ser el caso de instalar el simulador de la mano Shadow Hand simplemente se quiere probar y verificar que todo el software está correcto. Para ello, en lugar de programar, se recurre a un método más sencillo, realizar publicaciones puntuales de sondeo. Esto se realiza mediante el modificador *pub* de la orden *rostopic*, cuya estructura es la siguiente:

```
rostopic pub [topic] [msg_type] [args]
```

Un ejemplo de aplicación sería el siguiente:

```
$ rostopic pub -1 /turtle1/cmd_vel geometry_msgs/Twist -- '[2.0, 0.0,
0.0]' '[0.0, 0.0, 1.8]'
```

Mediante el cual se establece la velocidad lineal en 2.0 y la angular en 1.8.

Este es un ejemplo bastante complicado, así que echemos un vistazo a cada argumento en detalle.

Para ROS Hydro y versiones posteriores, se tiene que:

- Este comando publica mensajes en un topic determinado:

```
pub rostopic
```

- Esta opción obliga a realizar una única publicación

```
 -1
```

- Este es el nombre del topic en el que publicar:

```
/ Turtle1 / cmd_vel
```

- Este es el tipo de mensaje a utilizar cuando se publica:

```
geometry_msgs / Giro
```

- Esta opción (doble guión) le dice al intérprete de ROS que ninguno de los siguientes argumentos es una opción. De este modo es posible usar parámetros que requieran de dicho símbolo sin provocar confusiones, como puede ser la introducción de un número negativo.

```
-
```

- Los últimos parámetros hacen referencia a la estructura del mensaje. En este caso, `'[2,0, 0,0, 0,0]'` se convierte en el valor lineal con `x = 2,0` , `y = 0,0` , y `z = 0,0` , mientras que `'[0,0, 0,0, 1,8]'` es el valor angular con `x = 0,0` , `y = 0,0` , y `z = 1,8` .

```
'[2.0, 0.0, 0.0]' '[0.0, 0.0, 1.8]'
```

En la siguiente ilustración se encuentra un ejemplo de aplicación del procedimiento en la versión Groovy. Para más información consultar los manuales oficiales. [31]

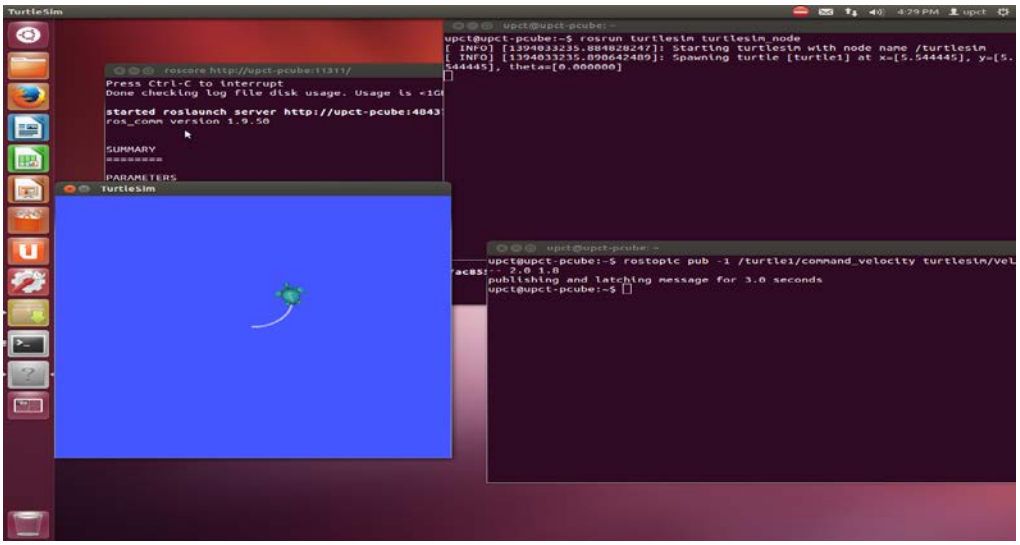



*Ilustración 60. Ejemplo de publicación por línea de comandos en ROS Groovy.*

En ciertas aplicaciones, por su carácter estricto, las publicaciones no pueden dejarse al azar, existiendo un tiempo máximo de comunicación que garantice la consecución de ciertos objetivos de tiempo real. En estos casos puede resultar interesante controlar el tiempo de publicación mediante el comando *hz rostopic*:

```
$ Hz rostopic / turtle1 / pose
```

Un ejemplo de aplicación sería el mostrado a continuación:

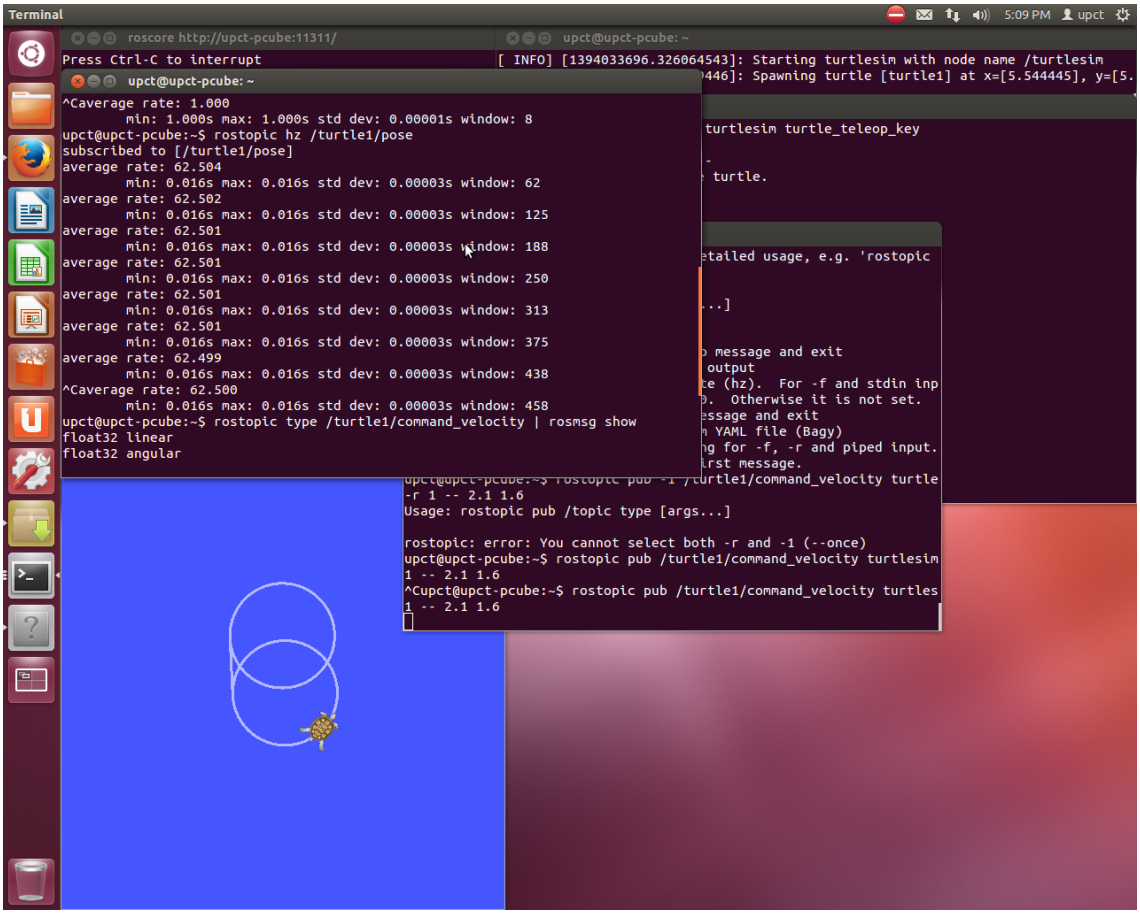



*Ilustración 61. Control del tiempo de publicación: comando hz.*

Vemos que la salida del comando hace alusión no sólo al tiempo de respuesta actual, sino que también contempla los márgenes que deben cumplirse para satisfacer la comunicación.

Para terminar este apartado, qué mejor que mostrar algunas gráficas de la evolución de nuestro sistema. Dada la velocidad que puede tener una tortuga, para este ejemplo, serán representadas las posiciones que alcanza a lo largo del tiempo:

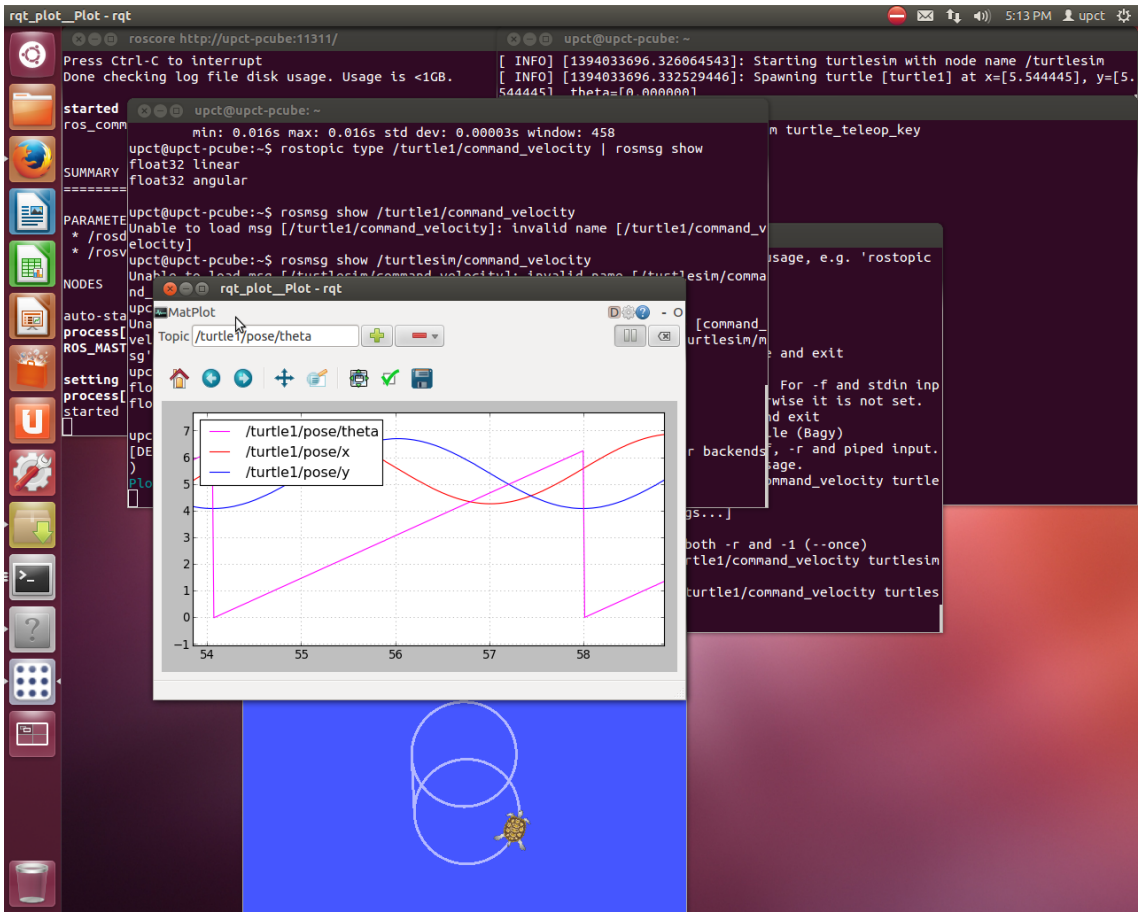



*Ilustración 62. Ejemplo de uso de rqt_plot para representar información.*

Así, en el ejemplo, se han mostrado las gráficas referidas a las posiciones x e y, junto al ángulo de giro instantáneo. Por los resultados, se comprueba que, efectivamente, la tortuga está generando trayectorias cerradas de forma circular.

## 5.7. Parámetros y servicios de ROS.

Hasta el momento se ha especificado como única forma de comunicación el envío y recepción de mensajes mediante el uso de topics. Sin embargo, conforme se avanza en el manejo del sistema e incrementa la complejidad de las aplicaciones, se alcanzan situaciones que requieren de una trama de comunicación más complicada. En estos desarrollos no es sólo necesario enviar un comunicado, sino que; el nodo emisor espera una respuesta por parte del receptor. Aunque esta implementación puede resolverse mediante el uso de mensajes de forma relativamente sencilla, se introducen retardos y tiempos de procesado innecesarios que pueden acarrear problemas de funcionamiento en ciertas aplicaciones. Además se realiza un uso de ROS "trampa" que no es del todo indicado. No obstante, en numerosos casos, cuidadosamente estudiados, podrán sustituirse los servicios por una agrupación de publicadores y subscriptores de diversos topics.

Dicho lo cual, toca presentar a los Servicios de ROS. Los Servicios son otra forma que poseen los nodos para comunicarse entre sí. Adicionalmente, dichos servicios permiten a los nodos enviar una solicitud y recibir una respuesta, tal y como se ha comentado en el párrafo anterior.

Al igual que cuando se trabaja con mensajes se dispone de un comando para consultar información que pueda resultar de interés, para los servicios, se emplea la orden *rosservice*. De este modo, si se quieren ver los servicios en ejecución, al igual que ocurría con su homónimo:

```
$ rosservice List
```

En caso de tener sólo iniciados el simulador de turtlebot y el panel de teleoperación, se obtendría algo similar a:

```
/clear
/kill
/reset
/rosout/get_loggers
```

```
/rosout/set_logger_level
/spawn
/teleop_turtle/get_loggers
/teleop_turtle/set_logger_level
/turtle1/set_pen
/turtle1/teleport_absolute
/turtle1/teleport_relative
/turtlesim/get_loggers
/turtlesim/set_logger_level
```

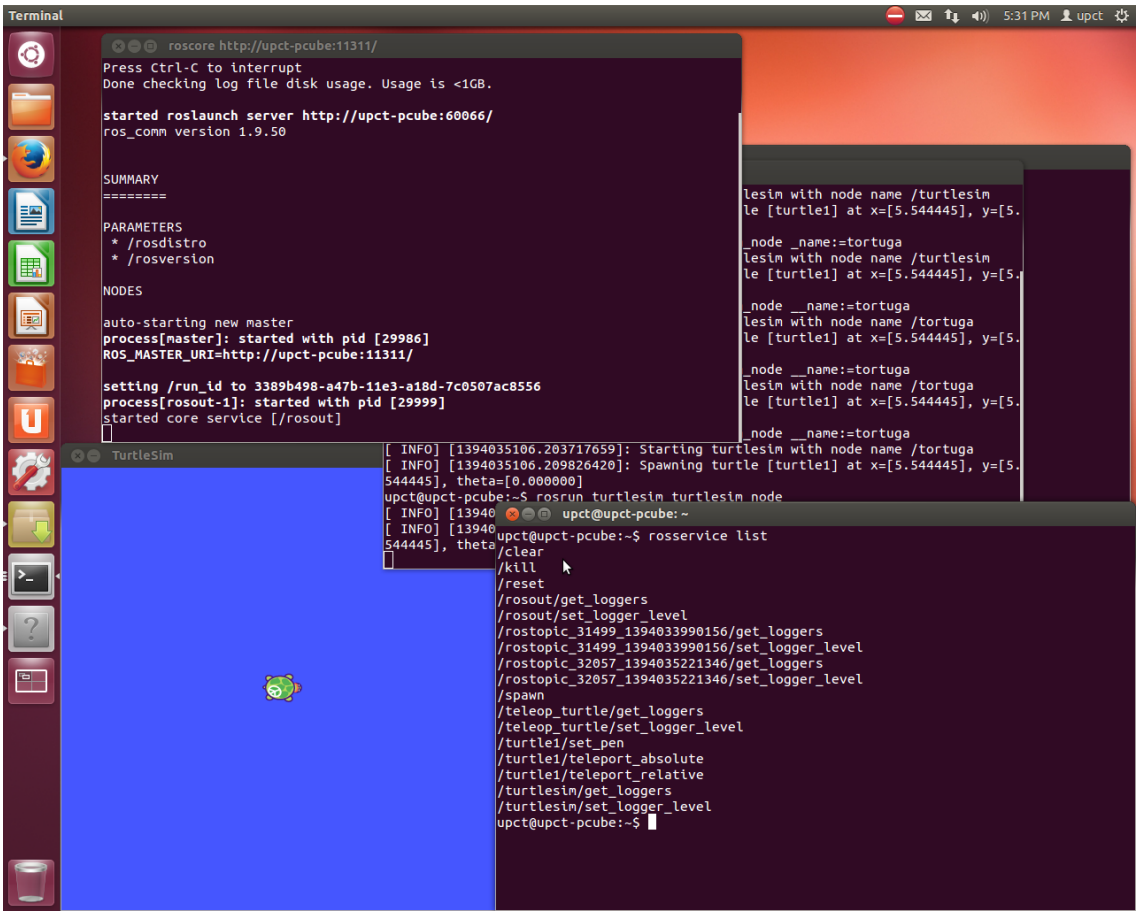



*Ilustración 63. Ejemplo de uso de Rosservice List.*

Una segunda analogía, como no podía ser menos, se encuentra en el tipo de datos a manejar por lo que también se distinguen los servicios, además de por su funcionalidad, en base al tipo de datos que manejan. Esta característica puede consultarse mediante el modificador *type*:

```
rosservice type [service]
```

La primera diferencia se haya en la sentencia de ejecución, en lugar de "correr" o ejecutar el servicio, se habla de "llamar a un servicio":

```
rosservice call [service] [args]
```

Así, si se busca crear una tortuga, se debe llamar a un servicio de creación de tortugas. Sin embargo, antes de proceder a ello, es necesario saber qué parámetros de entrada requiere el mismo para ser llamado:

```
$ rosservice type spawn| rossrv show
```

```
float32 x
float32 y
float32 theta
string name
---
string name
```

Se comprueba que son requeridas dos coordenadas que señalizan el lugar de aparición, un ángulo de giro y el nombre de la tortuga. Sin embargo, en caso de no proporcionar alguno de ellos, el servicio debería contar con parámetros por defecto.

```
$ rosservice call spawn 2 2 0.2 ""
```

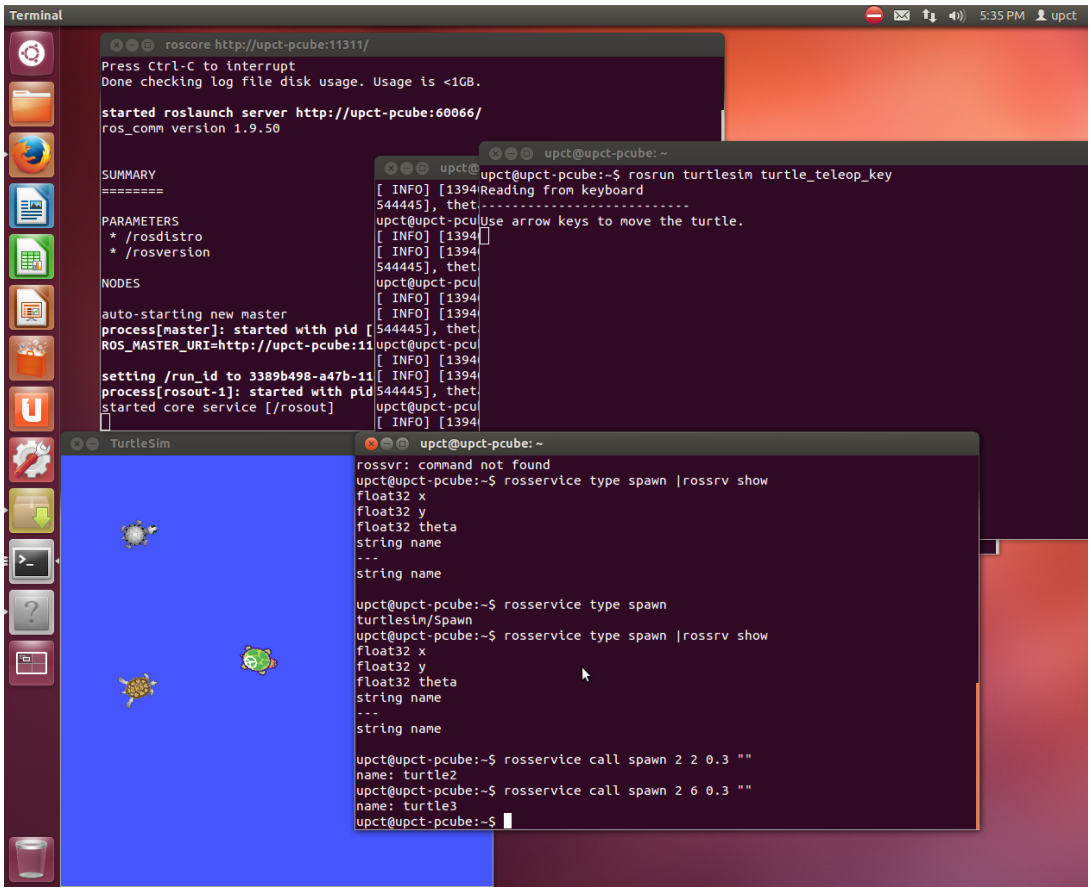



*Ilustración 64. Ejemplo de llamada a un servicio y cómo determinar la información a aportar.*

Tras ello se recibe un comando con el nombre de la nueva tortuga generada, al tiempo que una nueva tortuga aparece acompañando a la primera ya existente.

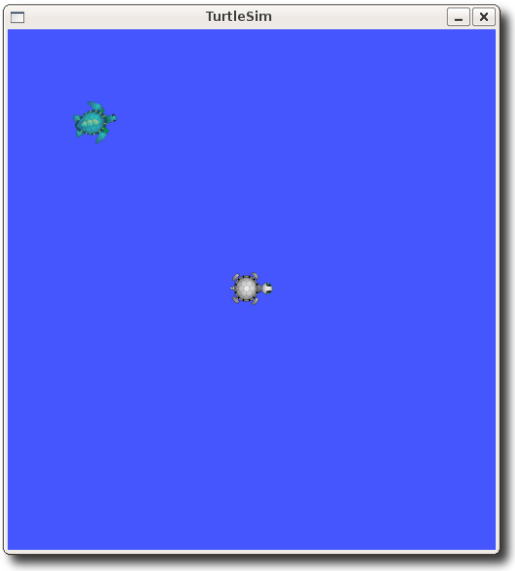



*Ilustración 65. Llamada a un servicio para crear un segundo turtlebot.*

Otro comando muy interesante a la hora de trabajar con servicios es *rosparam*. Éste permite almacenar y manipular información en el servidor de parámetros de ROS. Dicho servidor puede almacenar datos en distintos formatos entre los que se incluyen float, enteros… empleando para ello el lenguaje de sintaxis marcada YAML.

Para ver los parámetros que los servicios activos han almacenado en este servidor, se puede escribir en una ventana de terminal:

```
$ rosparam list
```

Puesto que sólo se tiene un servicio activo, la ventana del simulador, se comprueba que los únicos parámetros existentes son los que determinan el color del fondo de la aplicación:

```
/background_b
/background_g
/background_r
/roslaunch/uris/aqy:51932
/run_id
```

Otro dato interesante respecto a estos juegos de parámetros es la posibilidad de modificarlos mediante las opciones de sentencia set y get. Obsérvese la analogía con los métodos de manipulación de bases de datos como mySQL y Ajax:

```
rosparam set [param_name]
rosparam get [param_name]
```

De este modo, si escribiésemos:

```
$ rosparam set background_r 150
```

El color del fondo de la ventana en la que se encuentra turtlesim cambiaría:

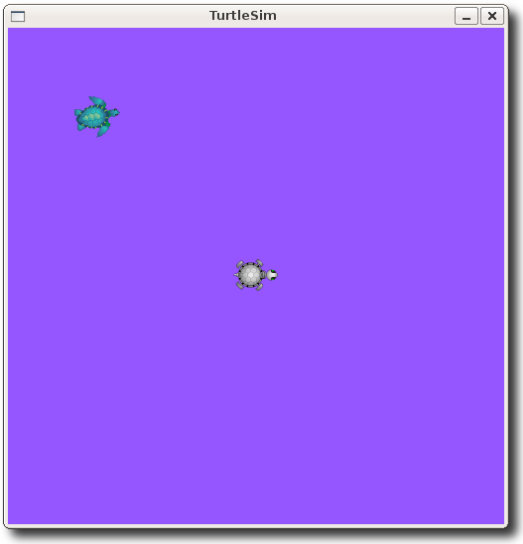



*Ilustración 66. Ejemplo de uso del comando rosparam.*

La opción get, por el contrario, devuelve la configuración de ese parámetro.

## 5.8. Ejecución de programas y visualización de datos: roslaunch y rqt_graph.

Aunque se ha empleado el paquete rqt previamente para visualizar las posiciones que alcazaba la tortuga a lo largo del tiempo, existen muchas más funcionalidades. En concreto, se mostrarán, a lo largo de este punto, las opciones de la GUI de rqt que permiten, entre otros aspectos, visualizar, de forma más estética y posiblemente intuitiva, los mensajes de log e información relativa al estado de los nodos.

Para ello, en primer lugar, se deben instalar los paquetes necesarios:

```
$ sudo apt-get install ros-<distro>-rqt ros-<distro>-rqt-common-plugin
s ros-<distro>-turtlesim
```

La inicialización de esta consola se debe realizar en modo *verbose*. Tras arrancar la interfaz de rqt, se procede a la lectura de los distintos avisos:

```
$ rosrun rqt_console rqt_console
$ rosrun rqt_logger_level rqt_logger_level
```

Una vez ejecutado, debería abrirse una nueva ventana con un aspecto similar al de la siguiente figura:

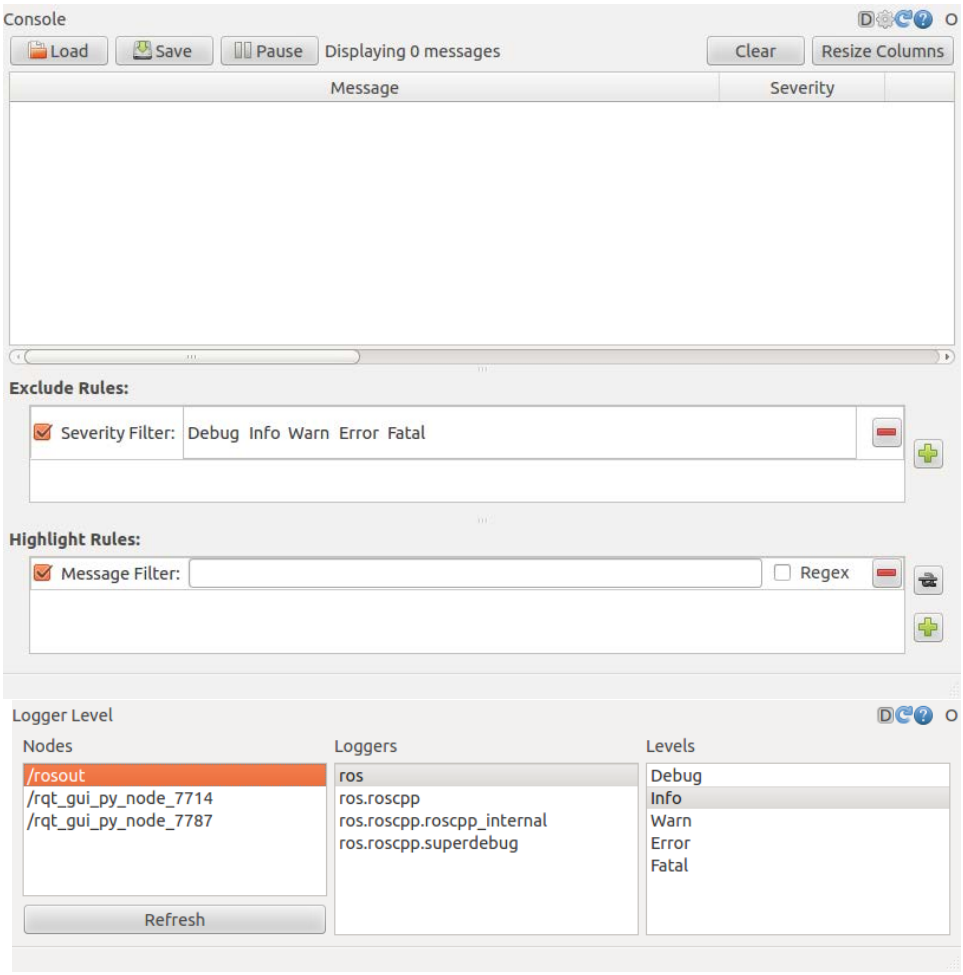



*Ilustración 67. Rqt_console con lectura de actividad.*

Conforme se inicien nodos y procesos, éstos irán apareciendo en la lista de nodos activos. Una vez seleccionado el nivel de los mensajes que se quieren monitorizar, en el cuadro de texto superior, se verá la actividad de todos los nodos marcados, según el nivel de profundidad escogido.

Respecto a los niveles de mensajes, deben realizarse ciertas matizaciones. Como se puede observar, se distingue entre cinco niveles de aviso o prioridad del mensaje, siendo el nivel más alto FATAL y el escalafón de menor importancia, DEBUG. Al seleccionar un nivel, se consideran todos los mensajes de prioridad igual o inferior.

Hasta este momento se han realizado ejecuciones de programas de carácter mono-nodal. No obstante, algunos de ellos engloban la ejecución de varios nodos. Cuando es necesario ejecutar más de un nodo con objeto de obtener un programa totalmente funcional se recurre a los ficheros de tipo launch. La herramienta *roslaunch* permite ejecutar scripts de tipo launch, que evitan emplear el comando *rosrun* iterativamente. Aparte de permitir ejecutar varios nodos simultáneamente, los usos de roslaunch y el abanico de posibilidades que éste despliega son tales que podría destinarse un TFG a analizar los entresijos del mismo. Una muestra de este potencial se puede observar en el código del simulador desarrollado para Hydro, que involucra los robots de shadow y un sensor Kinect ( *Capítulo 6. Software Desarrollado.*).

Para más información sobre este último tema, consultar la página oficial y direccionamientos que ésta realiza. [32]

31 http://wiki.ros.org/ROS/Tutorials/UsingRqtconsoleRoslaunch

# Anexo 6. Introducción al uso del simulador de Shadow.

Antes de comenzar, señalar que este apartado redunda ciertas imágenes y presenta carencia de algunas otras ya que en el momento de su realización el simulador de Shadow no podía emplearse como tal, debido a un problema surgido con la última actualización. Sin embargo, por las conversaciones mantenidas con Armando, se aprovecha este momento para darle las gracias por toda su ayuda, el simulador recuperará su funcionalidad en unas semanas.

Como se ha podido observar en numerosas ocasiones a lo largo del presente documento, Shadow integra, entre sus paquetes de ROS, un simulador para la herramienta de simulación Gazebo.

Tal y como señalan los desarrolladores de gazebo [33]:

"Un *simulador bien diseñado y correctamente desarrollado hace posible, entre otras tareas, probar algoritmos rápidamente, diseñar robots y realizar tests de actuación especial en entornos reales. Gazebo ofrece la habilidad de simular, de forma precisa y eficiente, complejos entornos exteriores e interiores para un amplio conjunto de robots. Y, por último, lo mejor de gazebo es que es de código libre y consta de una fervorosa y entusiasmada comunidad.*"

Tras este alarde de funcionalidades y ventajas, confirmar que, hasta donde ha sido probado para este proyecto, Gazebo desempeña una gran labor de simulación, cubriendo, en muchos casos, funcionalidades que no se esperaban de él.

Para ejecutar, el simulador de Shadow, existen tres opciones ya que este está compuesto por dos robots que suelen trabajar al unísono: Shadow arm y Shadow Hand. Por comodidad, y dado que en la realidad práctica de este proyecto la mano se encontraba adherida físicamente a un brazo de la marca robotnik

32 HTTP://WWW.GAZEBOSIM.ORG

LWA4P, se ha empleado el simulador completo. Así, ejecutando el archivo .launch para crear la escena correspondiente:

```
$ roslaunch sr_hand gazebo_arm_and_hand.launch
```

Si sólo se desease uno de los robots, bastaría con eliminar la parte del nombre que se refiere al robot no requerido.

Una vez la ventana del simulador haya aparecido, se debe comprobar que el robot ha adoptado alguna pose inicial. Además, si todo ha ido bien, se podrán mover las distintas articulaciones mediante publicaciones en los distintos topics que genera el simulador al ser lanzado.

Es en este momento, cuando el usuario reciente de ROS se ilusiona momentáneamente al comprobar la facilidad de uso. Sin embargo, poco tarda en cerciorarse de que desconoce los topics existentes, y en muchos casos el comando a emplear. Por suerte, ambos problemas se resuelven fácilmente.

Para publicar en un cierto topic se emplea la orden *rostopic pub*, tal y como se señaló en el apartado correspondiente del *Anexo 5. Primeros Pasos en ROS.* ( *5.6 Introducción al uso de topics.*). Como ejemplo, se desplazará la articulación 3 del dedo índice (en radianes) y, a continuación, descenderá el brazo.

```
$ rostopic pub /sh_ffj3_position_controller/command std_msgs/Float64 1
.5
$ rostopic pub /sa_ss_position_controller/command std_msgs/Float64 0.5
```

## 6.1. Lista de topics generados por el simulador.

El segundo inconveniente es solventado mediante otro modificador u opción del comando anterior, *rostopic*. Puesto que se busca una forma de obtener los topics que controlan la mano, qué mejor forma para ello que listar todos los topics activos cuando sólo se encuentra el simulador en ejecución:

```
$ rostopic list
```

Obteniéndose una lista como la siguiente:

rostopic list

/clock

/contacts/ff/distal

/contacts/ff/knuckle

/contacts/ff/middle

/contacts/ff/proximal

/contacts/lf/distal

/contacts/lf/knuckle

/contacts/lf/metacarpal

/contacts/lf/middle

/contacts/lf/proximal

/contacts/mf/distal

/contacts/mf/knuckle

/contacts/mf/middle

/contacts/mf/proximal

/contacts/palm

/contacts/rf/distal

/contacts/rf/knuckle

/contacts/rf/middle

/contacts/rf/proximal

/contacts/th/base

/contacts/th/distal

/contacts/th/hub

/contacts/th/middle

/contacts/th/proximal

/gazebo/link_states

/gazebo/model_states

/gazebo/parameter_descriptions

/gazebo/parameter_updates

/gazebo/set_link_state

/gazebo/set_model_state

/joint_states

/mechanism_statistics

/r_arm_cartesian_pose_controller/command

/r_arm_cartesian_pose_controller/state/error

/r_arm_cartesian_pose_controller/state/pose

/r_arm_joint_trajectory_controller/command

/r_arm_joint_trajectory_controller/gains/ElbowJRotate/parameter_descriptions

/r_arm_joint_trajectory_controller/gains/ElbowJRotate/parameter_updates

/r_arm_joint_trajectory_controller/gains/ElbowJSwing/parameter_descriptions

/r_arm_joint_trajectory_controller/gains/ElbowJSwing/parameter_updates

/r_arm_joint_trajectory_controller/gains/ShoulderJRotate/parameter_descriptions

/r_arm_joint_trajectory_controller/gains/ShoulderJRotate/parameter_updates

/r_arm_joint_trajectory_controller/gains/ShoulderJSwing/parameter_descriptions

/r_arm_joint_trajectory_controller/gains/ShoulderJSwing/parameter_updates

/r_arm_joint_trajectory_controller/gains/WRJ1/parameter_descriptions

/r_arm_joint_trajectory_controller/gains/WRJ1/parameter_updates

/r_arm_joint_trajectory_controller/gains/WRJ2/parameter_descriptions

/r_arm_joint_trajectory_controller/gains/WRJ2/parameter_updates

/r_arm_joint_trajectory_controller/state

/rosout

/rosout_agg

/sa_er_position_controller/command

/sa_er_position_controller/state

/sa_es_position_controller/command

/sa_es_position_controller/state

/sa_sr_position_controller/command

/sa_sr_position_controller/state

/sa_ss_position_controller/command

/sa_ss_position_controller/state

/sh_ffj0_position_controller/command

/sh_ffj0_position_controller/max_force_factor

/sh_ffj0_position_controller/pid/parameter_descriptions

/sh_ffj0_position_controller/pid/parameter_updates

/sh_ffj0_position_controller/state

/sh_ffj3_position_controller/command

/sh_ffj3_position_controller/max_force_factor

/sh_ffj3_position_controller/pid/parameter_descriptions

/sh_ffj3_position_controller/pid/parameter_updates

/sh_ffj3_position_controller/state

/sh_ffj4_position_controller/command

/sh_ffj4_position_controller/max_force_factor

/sh_ffj4_position_controller/pid/parameter_descriptions

/sh_ffj4_position_controller/pid/parameter_updates

/sh_ffj4_position_controller/state

/sh_lfj0_position_controller/command

/sh_lfj0_position_controller/max_force_factor

/sh_lfj0_position_controller/pid/parameter_descriptions

/sh_lfj0_position_controller/pid/parameter_updates

/sh_lfj0_position_controller/state

/sh_lfj3_position_controller/command

/sh_lfj3_position_controller/max_force_factor

/sh_lfj3_position_controller/pid/parameter_descriptions

/sh_lfj3_position_controller/pid/parameter_updates

/sh_lfj3_position_controller/state

/sh_lfj4_position_controller/command

/sh_lfj4_position_controller/max_force_factor

/sh_lfj4_position_controller/pid/parameter_descriptions

/sh_lfj4_position_controller/pid/parameter_updates

/sh_lfj4_position_controller/state

/sh_lfj5_position_controller/command

/sh_lfj5_position_controller/max_force_factor

/sh_lfj5_position_controller/pid/parameter_descriptions

/sh_lfj5_position_controller/pid/parameter_updates

/sh_lfj5_position_controller/state

/sh_mfj0_position_controller/command

/sh_mfj0_position_controller/max_force_factor

/sh_mfj0_position_controller/pid/parameter_descriptions

/sh_mfj0_position_controller/pid/parameter_updates

/sh_mfj0_position_controller/state

/sh_mfj3_position_controller/command

/sh_mfj3_position_controller/max_force_factor

/sh_mfj3_position_controller/pid/parameter_descriptions

/sh_mfj3_position_controller/pid/parameter_updates

/sh_mfj3_position_controller/state

/sh_mfj4_position_controller/command

/sh_mfj4_position_controller/max_force_factor

/sh_mfj4_position_controller/pid/parameter_descriptions

/sh_mfj4_position_controller/pid/parameter_updates

/sh_mfj4_position_controller/state

/sh_rfj0_position_controller/command

/sh_rfj0_position_controller/max_force_factor

/sh_rfj0_position_controller/pid/parameter_descriptions

/sh_rfj0_position_controller/pid/parameter_updates

/sh_rfj0_position_controller/state

/sh_rfj3_position_controller/command

/sh_rfj3_position_controller/max_force_factor

/sh_rfj3_position_controller/pid/parameter_descriptions

/sh_rfj3_position_controller/pid/parameter_updates

/sh_rfj3_position_controller/state

/sh_rfj4_position_controller/command

/sh_rfj4_position_controller/max_force_factor

/sh_rfj4_position_controller/pid/parameter_descriptions

/sh_rfj4_position_controller/pid/parameter_updates

/sh_rfj4_position_controller/state

/sh_thj1_position_controller/command

/sh_thj1_position_controller/max_force_factor

/sh_thj1_position_controller/pid/parameter_descriptions

/sh_thj1_position_controller/pid/parameter_updates

/sh_thj1_position_controller/state

/sh_thj2_position_controller/command

/sh_thj2_position_controller/max_force_factor

/sh_thj2_position_controller/pid/parameter_descriptions

/sh_thj2_position_controller/pid/parameter_updates

/sh_thj2_position_controller/state

/sh_thj3_position_controller/command

/sh_thj3_position_controller/max_force_factor

/sh_thj3_position_controller/pid/parameter_descriptions

/sh_thj3_position_controller/pid/parameter_updates

/sh_thj3_position_controller/state

/sh_thj4_position_controller/command

/sh_thj4_position_controller/max_force_factor

/sh_thj4_position_controller/pid/parameter_descriptions

/sh_thj4_position_controller/pid/parameter_updates

/sh_thj4_position_controller/state

/sh_thj5_position_controller/command

/sh_thj5_position_controller/max_force_factor

/sh_thj5_position_controller/pid/parameter_descriptions

/sh_thj5_position_controller/pid/parameter_updates

/sh_thj5_position_controller/state

/sh_wrj1_position_controller/command

/sh_wrj1_position_controller/max_force_factor

/sh_wrj1_position_controller/pid/parameter_descriptions

/sh_wrj1_position_controller/pid/parameter_updates

/sh_wrj1_position_controller/state

/sh_wrj2_position_controller/command

/sh_wrj2_position_controller/max_force_factor

/sh_wrj2_position_controller/pid/parameter_descriptions

/sh_wrj2_position_controller/pid/parameter_updates

/sh_wrj2_position_controller/state

/tf

/tf_static

De observar esta lista, se concluye que la primera parte de los temas pertenecen a elementos del brazo, mientras que la segunda parte de ellos hace referencia a distintas opciones de comunicación con el robot manipulador. Se puede comprobar, además, la cantidad de topics con los que se relaciona cada articulación. Sin embargo, a pesar de esta aglomeración, para controlar la mano, principalmente se emplean aquellos relacionados con los actuadores de posición (/sh_rfj0_position_controller/command), sensores de posición (/sh_rfj0_position_controller/state), sensores de presión (/contacts/rf/distal) y, puntualmente, actuadores de velocidad (/sh_rfj0_velocity_controller/command).

*NOTA: los controles de velocidad sólo están disponibles en ciertas versiones. Sin embargo, aunque no aparezcan para su control directo, es posible emplearlos al programar especificando ciertas estructuras temporales. No se profundizará en estos aspectos ya que el objeto del proyecto no trata de abordar temas de manipulación dinámica.*

*Con estos datos y consultando los ejemplos existentes en el paquete shadow_robot, que puede ser encontrado mediante la herramienta rospack, es posible iniciarse en la creación de programas.*

## 6.2. Esquemático de la mano Shadow Hand para gazebo.

A la hora de crear el modelo urdf que emplea gazebo se ha considerado un robot Shadow Hand que coincide con el del siguiente esquemático:

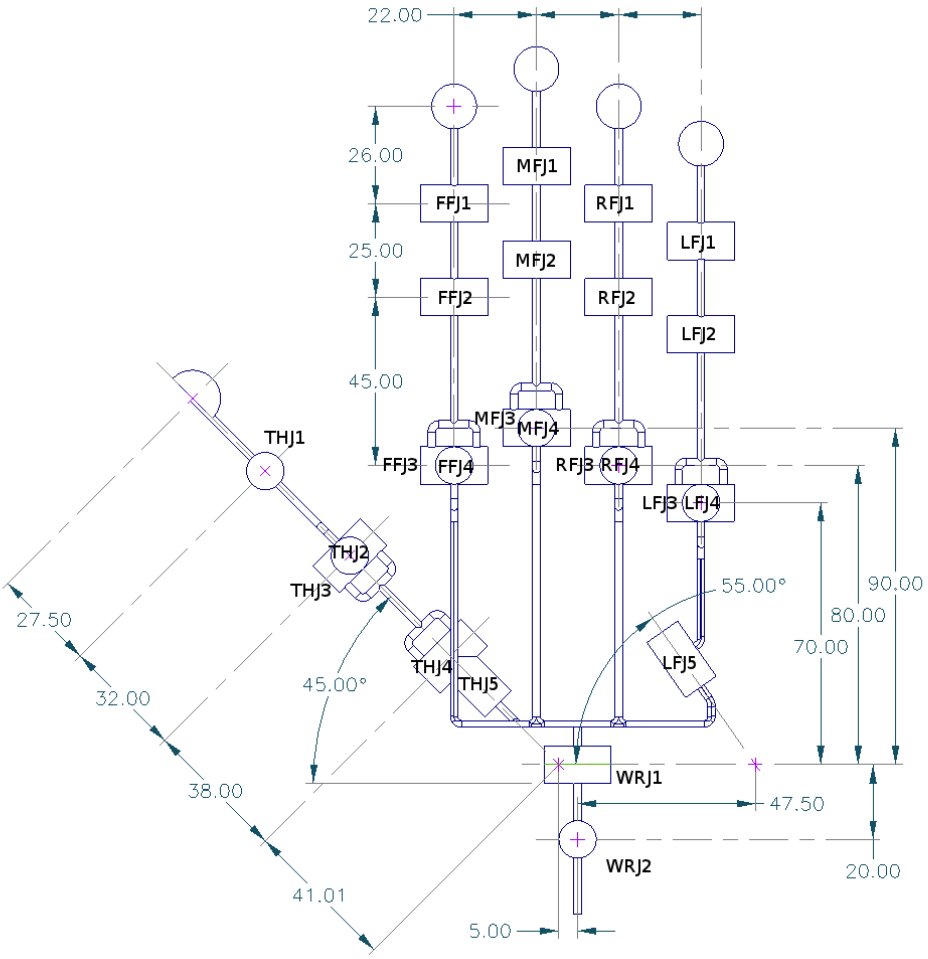



*Ilustración 68. Esquemático de Shadow Hand para gazebo.*

Imagen facilitada por Shadow Company, que puede ser encontrada en el apartado reservado para la empresa en la página oficial de ROS. [Shadow Hand Gazebo]. [34]

33 [Shadow Hand Gazebo] http://wiki.ros.org/shadow_robot?action=AttachFile&do=view&target=annotated_hand.png

# Anexo 7. Instalación de PCL y drivers necesarios.

## 7.1. Introducción y consejos previos.

El presente anexo tiene como objetivo cubrir todos los pasos a realizar para llevar a cabo de forma satisfactoria el proceso de instalación de PCL. Al igual que ocurre con otro tipo de software, librerías… PCL presenta distintas versiones, en el momento en que se desarrolló el presente proyecto la versión estable era la 1.7, por ello todos los comandos están orientados al uso de esta versión de librerías.

Antes de comenzar, señalar que ROS, durante su proceso de instalación, incluye las librerías de PCL. No obstante, al depender de la versión disponible en el momento de lanzamiento de la distribución de ROS utilizada, éstas no siempre son las deseadas. Para el caso de ROS *Hydro*, no es necesario realizar ninguna instalación ya que integra *PCL 1.7-version*, sin embargo, no ocurre lo mismo con Groovy. De este modo, al seguir el proceso de instalación, con Groovy presente en el equipo objetivo, se tendrán problemas de acceso a librerías dado que se dispondrá de dos versiones de PCL instaladas. En este caso, puesto que la instalación se suele realizar únicamente en caso de requerirse una versión posterior, ésta será la que siempre se utilizará. De este modo, la forma más sencilla de evitar dicho problema de librerías consiste en obligar al sistema, en el CmakeFile de compilación, a emplear la versión más actualizada para la creación de los archivos ejecutables. Para ello, basta con escribir al inicio del archivo Cmake: REQUIRED_VERSION 1.7.

Una vez realizadas estas consideraciones y proporcionada solución al problema más usual, es momento de dar comienzo al proceso de instalación.

## 7.2. Instalación de PCL.

El proceso de instalación puede ser encontrado en las páginas oficiales de ROS y PCL así como en multitud de sitios de terceros. Desde aquí, se recomienda seguir el proceso de instalación, configuración y puesta en marcha de PCL, junto a una cámara de profundidad, que se encuentra descrito, detalladamente, en [Robótica Unileon] [35].

A continuación, se abordará el proceso de instalación por SCV convencional, en caso de querer realizarse una instalación *"desde fuente"* dirigirse a la página citada.

Para instalar la versión 1.7 de PCL en nuestro sistema, el primer paso a realizar consiste en buscar las librerías de PCL y openni (driver para cámaras de profundidad) en nuestro sistema, solicitando su adquisición en caso de no ser éstas detectadas:

```
sudo apt-get install libpcl-1.7-all libpcl-1.7-all-dev libopenni-de
v libopenni-sensor-primesense-dev -y
```

Dado que ROS ya está instalado, se realiza la configuración PPA de repositorios siguiente:

```
sudo add-apt-repository ppa:v-launchpad-jochen-sprickerhof-de/pcl
sudo apt-get update
sudo apt-get install build-essential cmake libpcl-all libpcl-all-de
v -y
```

Una vez ejecutados estos comandos y aceptadas todas las confirmaciones solicitadas, se tendrá instalado todo el soporte software necesario para adquirir imágenes mediante un sistema de visión con detección de profundidad y procesar éstas mediante técnicas de nubes de puntos.

34 [ Robotíca Unileon] http://robotica.unileon.es/mediawiki/index.php/PCL/OpenNI_tutorial_1:_Installing_and_testing

Una sencilla forma de realizar una prueba de la instalación consiste en ejecutar el visualizador facilitado por Robótica Unileon, en la dirección antes citada, que muestra, en tiempo real, el entorno capturado por la cámara.

## 7.3. Instalaciones adicionales: FLANN y HDF5.

Al realizar la instalación de PCL, por defecto, se adquiere y configura la versión estable tal y como se ha hecho hasta el momento. No obstante, cada versión en desarrollo contempla una parte experimental que puede ser instalada parcial o totalmente. Así, para emplear ciertas funciones específicas de PCL es necesario instalar y/o configurar algunas librerías adicionales. Concretamente, en este momento, para procesar modelos vfh y realizar determinados procesos de matching se requiere de ciertas funcionalidades experimentales: cargar la funcionalidad FLANN y librerías HDF5.

### 7.3.1. Adquisición de FLANN.

Las librerías FLANN (Fast Library for Approximate Nearest Neighbours) se emplean en la obtención de los descriptores de modelos VFH. La función de estos descriptores es caracterizar los clusters de objetos (conjunto de puntos constituyente, por sí mismo, de una nube que define íntegra e inequívocamente un objeto) que posibilitan llevar a cabo tareas de reconocimiento. A día de hoy, las librerías de FLANN se instalan como parte experimental probada. Esto quiere decir que, aunque presentes en el sistema, no están habilitadas para ser usadas. Por ello, todos los archivos que precisen de esta funcionalidad, deberán contar con un segundo CMake en el directorio denominado findFLANN.cmake.

Este archivo puede ser descargado de la cuenta del proyecto pcl en github:

*https://github.com/otherlab/pcl/blob/master/cmake/Modules/FindFLANN.cmake*

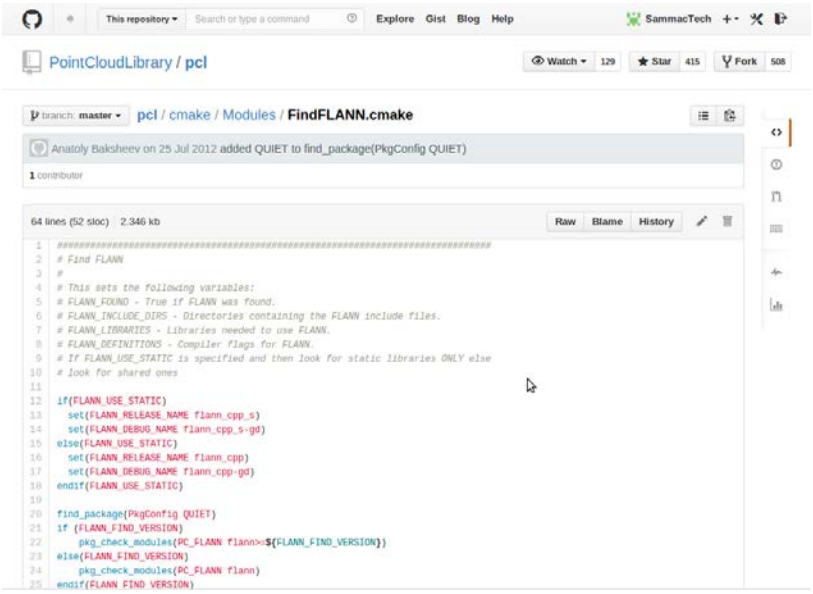



*Ilustración 69. CMake para uso de FLANN.*

Debiendo ser guardado en la carpeta principal del programa que se está creando, tal y como se muestra en la imagen siguiente:

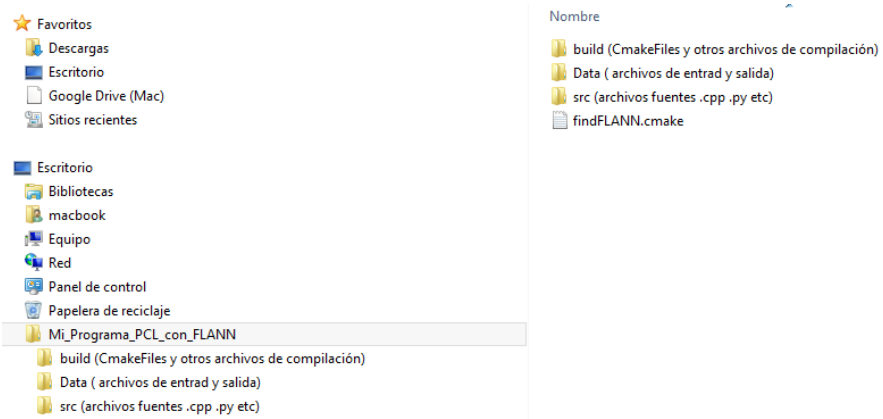



*Ilustración 70. Ejemplo de uso de FLANN.*

Una vez realizado este proceso ya es posible usar esta funcionalidad, en este paquete concreto. Como se puede deducir, cada uno de los ejecutables a crear que precise de FLANN requerirá repetir estos pasos.

### 7.3.2. Instalación de librerías auxiliares HDF5.

Las librerías HDF5 permiten, entre otros aspectos, emplear ciertos formatos de datos y funciones de tratamiento de la información para trabajar con modelos vfh.

Según apuntan sus desarrolladores:

"HDF5 es un modelo de datos, librerías y formatos de ficheros para almacenar y manejar la información. Esta tecnología da soporte a una variedad de tipos de datos ilimitada habiendo sido diseñada para ser flexible y eficiente desde el punto de vista de manejo de archivos de gran complejidad y tamaño. HDF5 es portable y ampliable, permitiendo que sea empleado en diversas aplicaciones." [36]

A diferencia de FLANN, estas librerías han de ser instaladas. En lugar de acudir al terminal y recurrir a la pertinente ejecución de comandos, en este caso, se optará por emplear el gestor de paquetes synaptic. Por medio de este gestor, para usuarios avanzados, se facilita la instalación de paquetes individuales complementarios a un software ya instalado. Como se puede observar en la imagen, tras abrir el gestor de paquetes (ya sea haciendo click en el icono del dash o bien navegando por las aplicaciones disponibles, primer icono del dash en la parte superior izquierda) se escribe en término de buqueda "hdf5". Se comprobará que; al pinchar la opción *buscar*, se despliega una abundante lista con todos los paquetes disponibles de esta tecnología.

Puesto que sólo necesitamos los paquetes que permiten emplear el tratamiento de datos y definen distintos formatos de datos, en lugar de seleccionar la suite completa, se deben incluir los paquetes mostrados:

- H5utils.
- Hdf5-tools
- Libhdf5-serial-dev

[35] HTTP://WWW.HDFGROUP.ORG/HDF5/

NOTA: En un cierto punto, el sistema preguntará por instalar algunos paquetes que son necesarios para el correcto funcionamiento de alguno de los seleccionados, deberá aceptarse esta sub-instalación.

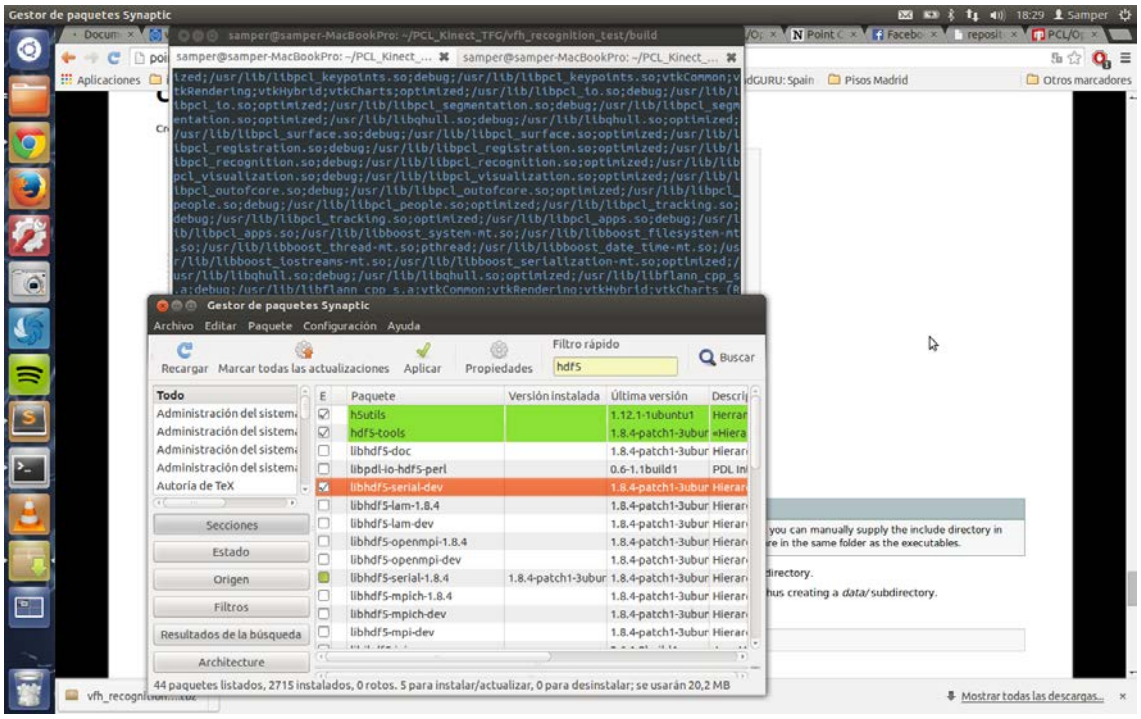



*Ilustración 71. Selección librerías HDF5 en synaptic.*

Una vez finalizados todos los pasos mostrados en este anexo se dispondrá de una instalación de pcl que permite realizar todos los tutoriales, y por ende, gran parte de las aplicaciones que se desee llevar a cabo.

# Anexo 8. Primeros Pasos con PCL.

Como se citó en el capítulo 5, sección *Primeros pasos con PCL*, este anexo tiene como finalidad orientar los primeros pasos mediante la adición de explicaciones sobre la estructura de PCL, tratando de servir como complemento a la relación de tutoriales oficiales. Señalar, además, que la comunidad de PCL, al igual que la de ROS, se caracteriza por la atención y colaboración entre sus miembros, existiendo así una dirección de mailing-list (http://www.pcl-users.org/) en la que se recomienda publicar cualquier duda o problema que surja ya que la atención y ayuda recibida ahorra una gran cantidad de horas de infructuosas búsquedas y dolores de cabeza.

## 8.1. Almacenamiento de la información y formas de representación.

Pasando a analizar PCL, el primer concepto a aclarar se remite a los posibles formatos de representación de la información tridimensional. Al trabajar con estructuras tridimensionales, se distingue, en función de la distribución de la información, entre representaciones organizadas y desorganizadas. Además, dichas representaciones se encuentran subdivididas, teniéndose:

- Representaciones desorganizadas:
  - Nubes de Puntos: Listas desorganizadas de vértices.
  - Malla poligonal: Lista desorganizada de vértices y sus relaciones o conexiones.
- Representaciones Organizadas:
  - Mapa binario o Voxel: rejilla tridimensional de valores de densidad.

- Imagen de rango: Rejilla bidimensional formada a partir de coordenadas tridimensionales.

También se diferencia entre dos tipos de datos tridimensionales:

- Datos Puramente Tridimensionales.
- Datos RGB-D: almacenan la intensidad de los tres canales junto a información sobre la geometría de la escena.

En la siguiente imagen, se pueden observar las diferencias entre cada una de las distintas formas de representación de la información, así como las transformaciones a realizar para pasar de una a otra:

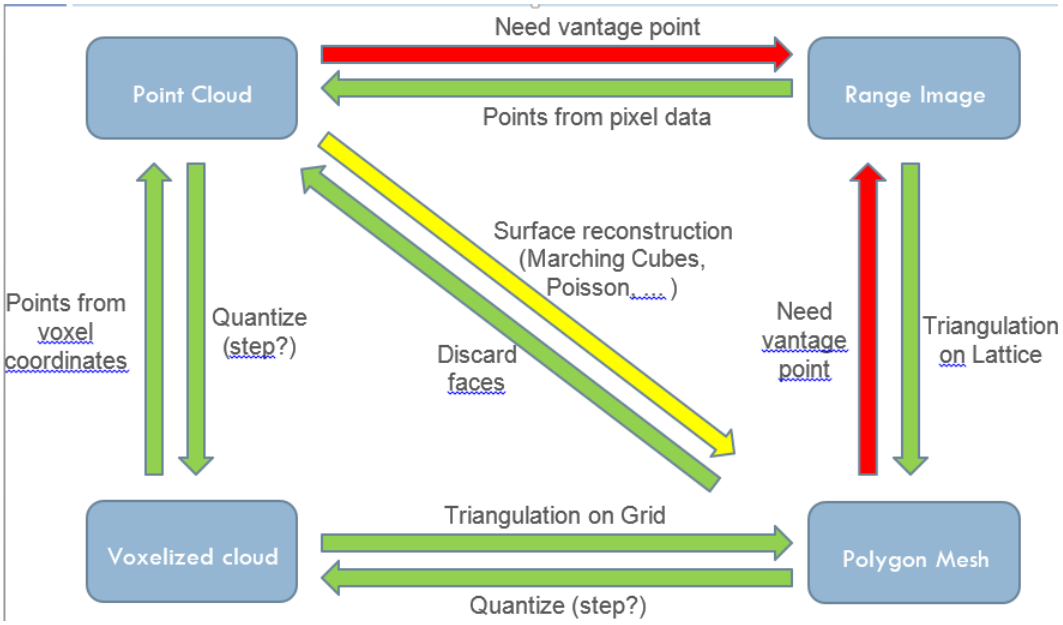



*Ilustración 72. PCL: Representación de datos 3D.*

En cuanto a PCL, éste puede trabajar con ambas formas de representación de la información, siendo las nubes de puntos organizadas y desorganizadas almacenadas por un tipo de estructura específica de la librería: pcl::PointCloud, dentro de la cual se encuentran varios subtipos y clasificaciones como XYZPoints, XYZRGBPoints …

El único caso de tratamiento particular se presenta en el uso de estructuras de tipo Voxel, almacenadas por la misma estructura pero con funciones específicas de procesado.

## 8.2. Reconocimiento de objetos y representación de la información.

Generalmente, para el reconocimiento de objetos se emplean modelos en 3D obtenidos mediante software CAD o uso de escáneres de alta precisión. A pesar de adquirir y almacenar el modelo en 3D, el proceso de reconocimiento se realiza representando la información en modelos denominados 2.5D. Este sistema 2.5D se basa en enfrentar las distintas vistas del modelo contra la imagen de escena adquirida. Señalar que, a diferencia de los modelos, las escenas suelen capturarse con formato 2.5D ya que; las cámaras sólo son capaces de captar la parte del entorno a la que enfocan, siendo desconocido el resto del "*mundo*", esto es, zonas no captables de forma directa por el dispositivo como partes traseras de objetos, laterales…

Entre los distintos motivos que justifican el uso de la representación 2.5D destacan el ahorro de recursos y las dificultades que ofrece la conversión de 2.5D a 3D, en contraposición a la facilidad para obtener vistas 2.5D de un modelo tridimensional.

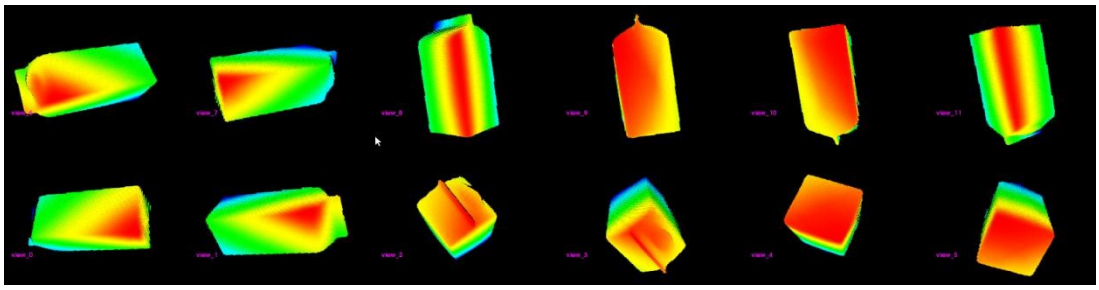


*Ilustración 73. Vistas en 2.5D de un modelo 3D.*

El proceso de reconocimiento e identificación de objetos presenta dos partes de realización, uno para cada grupo involucrado. En la siguiente imagen se pueden encontrar ambos caminos a seguir:

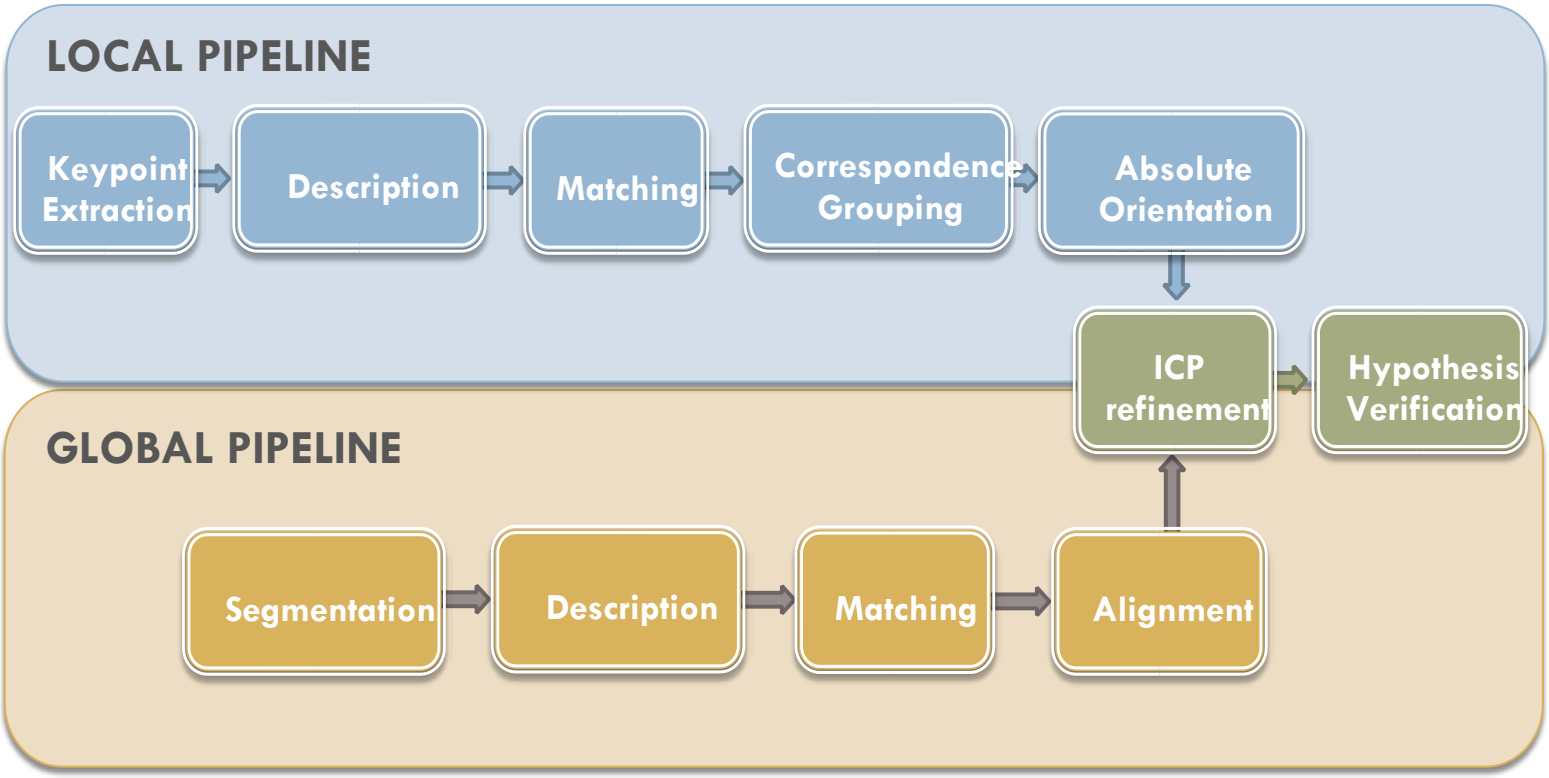



*Ilustración 74. Posibilidades para realizar reconocimiento de objetos.*

El camino local es el aplicado cuando se presentan varios objetos en la misma escena y/o la información no se encuentra organizada. De este modo, es necesario obtener los puntos clave de la escena (generalmente tres, para definir planos de puntos en la misma profundidad, conocidos a priori), para posteriormente aplicar una serie de técnicas que concluyen en la extracción de las nubes ordenadas de puntos que definen individualmente cada objeto.

Por otro lado, se encuentra el camino global, de aplicación a todas las escenas capturadas. Este proceso cubre los distintos aspectos que se han de aplicar para eliminar el ruido y planos sobrantes. Entre los planos sobrantes se encuentran aquellos de información irrelevante (una pared o la mesa sobre la que se apoyan los objetos, por ejemplo), los denominados inliners (puntos negros no

adquiridos por las limitaciones físicas de la cámara como puede ser la zona muerta del sensor)…

Señalar que los procesos de matching y correspondence grouping no realizan ninguna comparación entre la escena y el modelo, sino que forman parte de la fase de tratamiento de la información. Estando ambos destinados a localizar qué puntos de la información analizada se encuentran lo suficientemente próximos entre sí y cuentan con una profundidad suficiente como para ser parte del mismo objeto. Así, de juntar ambos flujos, se obtienen los objetos individuales, siendo posible aplicar un procedimiento de correspondencia como el empleado en el *tutorial de 6DOF Clustering de la página de PCL*.

# Anexo 9. Algoritmo cinemático programado.

La estrategia seguida a la hora de desarrollar el algoritmo cinemático se ha basado en considerar el comportamiento de todos los dedos análogo con objeto de reducir el número de posibles situaciones a calcular y grados de libertad manejados (algunas articulaciones responsables de ciertas abducciones no son consideradas, por ejemplo). Puesto que estas simplificaciones pueden suponer grandes limitaciones de agarre, la parte software complementaria desarrollada (funciones de ajuste, bucles de comprobación…) trata de apoyar al algoritmo y mejorar su comportamiento.

Sin más preámbulos, para el cálculo del algoritmo se parte de las siguientes consideraciones:

- Todos los dedos presentan un movimiento análogo ya que constan con el mismo número de falanges, aunque el pulgar difiere en orientación (corrección aplicada a nivel software). En altura, desde el extremo del índice al comienzo del pulgar, el centro se encuentra en la mediatriz de la recta que une ambos puntos.
- Únicamente se consideran tres grados de libertad en los dedos: dos de posición y uno de orientación.
- El centro de coordenadas se localiza en el lateral externo del índice a la altura en la que se produce la unión entre el metacarpo y la articulación proximal del citado dedo.
- La distancia a la que se encuentra el objeto de la mano será en todo momento, como mínimo, ligeramente superior a 0. Esta imposición se establece debido al número de términos sinusoidales barajados.

- Los objetos a coger presentarán simetría radial o de revolución.

Así, para cada dedo, se tendrá una situación como la siguiente:

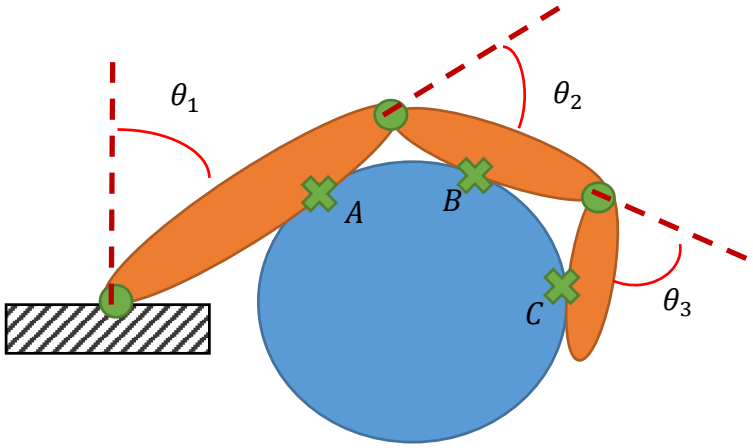



*Ilustración 75. Esquema 1 de cálculo del algoritmo cinemático.*

Según el esquema, los puntos A, B y C se definen como puntos tangenciales a la circunferencia. Así, el ángulo formado entre la línea horizontal que pasa por el centro de la circunferencia y el radio perpendicular a la tangente en A es $\theta_1$ debido al cruce de secantes.

De este modo, se llega a determinar que los ángulos de las articulaciones se encuentran contenidos en la propia circunferencia tal y como muestra la siguiente figura:

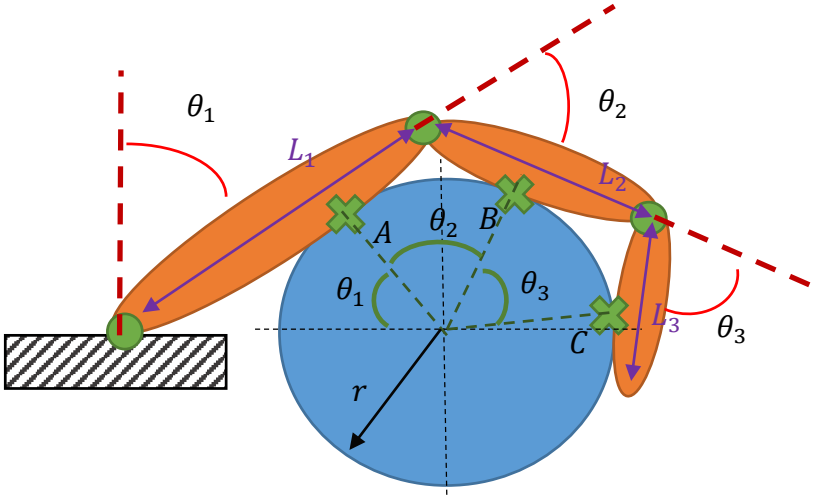



*Ilustración 76. Esquema 2 de cálculo del algoritmo cinemático.*

Las notaciones empleadas, a continuación, serán de la forma:

- $Coseno\ (\theta_1) = C_1$
- $Seno\ (\theta_1) = S_1$
- $Seno\ de\ una\ suma \rightarrow Seno\ (\theta_1 + \theta_2) = S_{12}$
- $Coeno\ de\ una\ suma \rightarrow Coseno\ (\theta_1 + \theta_2) = C_{12}$
- $Proyecciones\ en\ el\ eje\ X\ del\ punto\ A: A_x$
- $Proyecciones\ en\ el\ eje\ Y\ del\ punto\ A: A_y$
- $Radio\ de\ la\ circunferencia: r$
- $Longitud\ desde\ el\ origen\ al\ centro\ de\ la\ circunferencia: l_c$
- $Longitud\ desde\ el\ origen\ al\ punto\ A: a$

**Cálculo de $\boldsymbol{\theta_1}$ y a**.

$$\begin{cases} A_x = l_c - rC_1 = aS_1 \\ A_y = rS_1 = aC_1 \end{cases} \rightarrow \begin{cases} l_c = aS_1 + rC_1 & (I) \\ 0 = -rS_1 + aC_1 & (II) \end{cases}$$

*De hacer suma de cuadrados con el segundo conjunto*:

$$l_c{}^2 = a^2 + r^2 \rightarrow \boxed{a = \sqrt{l_c{}^2 - r^2}}$$

*Por otro lado, de realizar la suma de productos r*(I) +a*(II) y despejar, sustituyendo el valor de a calculado:*

$$\boxed{\theta_1 = \arccos\left(\frac{r}{l_c}\right)}$$

**Cálculo de $\boldsymbol{\theta_2}$ y b**.

$$\begin{cases} B_x = l_c - rC_{12} = l_1S_1 + bS_{12} \\ B_y = rS_{12} = l_1C_1 + bC_{12} \end{cases} \rightarrow \begin{cases} l_c - l_1S_1 = bS_{12} + rC_{12} & (I) \\ -l_1C_1 = -rS_{12} + aC_{12} & (II) \end{cases}$$

*De hacer suma de cuadrados con el segundo conjunto*:

$$\boxed{b = \sqrt{l_c{}^2 + l_1{}^2 - r^2 - 2l_cl_1S_1}}$$

*Por otro lado, de realizar la suma de productos r*(I) +b*(II) y despejar, sustituyendo el valor de b calculado:*

$$C_{12} = \frac{rl_c - rl_1S_1 - bl_1C_1}{r^2 + b^2}$$

*Desglosando la suma cosenoidal y despejando:*

$$\theta_2 = arccos\left[\frac{rl_c - rl_1S_1 - bl_1C_1}{r^2 + b^2}\right] - \theta_1$$

Y, por último, **Cálculo de $\boldsymbol{\theta_3}$ y c**.

$$\begin{cases} C_x = l_c - rC_{123} = l_1S_1 + l_2S_{12} + cS_{123} \\ C_y = rS_{123} = l_1C_1 + l_2C_{12} + cC_{123} \end{cases}$$

$$\rightarrow \begin{cases} l_c - l_1S_1 - l_2S_{12} = rC_{123} + cS_{123} & (I) \\ -l_1C_1 - l_2C_{12} = cC_{123} - rS_{123} & (II) \end{cases}$$

*De hacer suma de cuadrados con el segundo conjunto*:

$$c = \sqrt{\gamma^2 + \delta^2 - r^2}$$

*Donde:*

- $\gamma = l_c - l_1S_1 - l_2S_{12}$
- $\delta = -l_1C_1 - l_2C_{12}$

*Por otro lado, de realizar la suma de productos r*(I) +c*(II) y despejar, sustituyendo el valor de b calculado:*

$$C_{123} = \frac{r\gamma + c\delta}{r^2 + c^2}$$

*Desglosando la suma cosenoidal y despejando:*

$$\theta_3 = arccos\left[\frac{r\gamma + c\delta}{r^2 + c^2}\right] - \theta_2 - \theta_1$$

Recordar que estos ángulos son tratados en radianes, mientras que el robot Shadow Hand trabaja con grados sexagesimales.

# Anexo 10. Variabilidad de presión en los sensores de la Shadow Hand.

Uno de los principales problemas de que adolece la Shadow Hand disponible en el laboratorio se encuentra en los sensores de presión. Éstos presentan valores diferentes de carácter no lineal, ni modelable fácilmente, cada vez que se inicia el robot.

Para solucionar este inconveniente, se modificó el programa final para trabajar con valores de presión relativa, adquiriendo así mayor importancia los procesos de ajuste de presión y control antideslizamiento. Para establecer los valores de presión relativa de referencia se realizaron un conjunto de experimentos cuyos resultados se muestran en la siguiente tabla.

| Botella | Exp 1 | | Exp 2 | | Exp 3 | | Exp 4 | |
|---|---|---|---|---|---|---|---|---|
| | M.Vacia | Agarre | M.Vacia | Agarre | M.Vacia | Agarre | M.Vacia | Agarre |
| FF | 6.64 | 7.07 | 3.28 | 4.45 | 3.28 | 4.6 | 5.11 | 6.05 |
| RF | 0.74 | 0.77 | 0.36 | 0.43 | 0.36 | 0.53 | 0.59 | 0.6 |
| TH | 0.54 | 0.55 | 0.27 | 0.33 | 0.27 | 0.37 | 0.46 | 0.45 |

| APresion | Exp 1 | Exp 2 | Exp 3 | Exp 4 | Media |
|---|---|---|---|---|---|
| FF | 0.43 | 1.17 | 1.32 | 0.94 | 0.965 |
| RF | 0.03 | 0.07 | 0.17 | 0.01 | 0.07 |
| TH | 0.01 | 0.06 | 0.1 | 0.01 | 0.045 |

| CAJA | Exp 1 | | Exp 2 | | Exp 3 | | Exp 4 | |
|---|---|---|---|---|---|---|---|---|
| | M.Vacia | Agarre | M.Vacia | Agarre | M.Vacia | Agarre | M.Vacia | Agarre |
| FF | 3.28 | 4.29 | 3.28 | 4.25 | 6.64 | 7.56 | 4.96 | 5.89 |
| RF | 0.36 | 0.48 | 0.36 | 0.48 | 0.74 | 0.82 | 0.39 | 0.64 |
| TH | 0.277 | 0.35 | 0.277 | 0.37 | 0.54 | 0.6 | 0.487 | 0.49 |

| APresion | Exp 1 | Exp 2 | Exp 3 | Exp 4 | Media |
|---|---|---|---|---|---|
| FF | 1 | 0.97 | 0.92 | 0.93 | 0.96 |

| | | | | | |
|---|---|---|---|---|---|
| RF | 0.12 | 0.12 | 0.08 | 0.05 | 0.1 |
| TH | 0.073 | 0.043 | 0.06 | 0.01 | 0.05 |

| Ambientador | Exp 1 | | Exp 2 | | Exp 3 | | Exp 4 | |
|---|---|---|---|---|---|---|---|---|
| | M.Vacia | Agarre | M.Vacia | Agarre | M.Vacia | Agarre | M.Vacia | Agarre |
| FF | 4.88 | 4.92 | 4.80 | 4.85 | 6.64 | 6.67 | 4.96 | 5.01 |
| RF | 0.59 | 0.62 | 0.54 | 0.60 | 0.74 | 0.76 | 0.39 | 0.42 |
| TH | 0.48 | 0.5 | 0.45 | 0.5 | 0.54 | 0.57 | 0.487 | 0.52 |

| APresion | Exp 1 | Exp 2 | Exp 3 | Exp 4 | Media |
|---|---|---|---|---|---|
| FF | 0.04 | 0.05 | 0.03 | 0.05 | 0.04 |
| RF | 0.035 | 0.06 | 0.02 | 0.03 | 0.036 |
| TH | 0.04 | 0.05 | 0.03 | 0.033 | 0.04 |

En un principio, se emplearon los valores obtenidos como medias, aunque más adelante se reajustaron éstos a los valores que figuran en el archivo final de presiones estables:

```
Valores de presión estable para la agarrar la botella rosa (FF_touch, RF_touch, TH_Touch)

1.2

0.15

0.05

Valores de presión estable para la agarrar la caja (FF_touch, RF_touch, TH_Touch)

1.3

0.15

0.08

Valores de presión estable para la agarrar el ambientador (FF_touch, RF_touch, TH_Touch)

0.03

0.03

0.0
```

# Anexo 11. Proceso de Desarrollo de un simulador del robot junto a Kinect.

Para probar las posibilidades de controlar el robot desde ROS se empleó, principalmente el simulador proporcionado por Shadow. No obstante, alcanzado cierto punto del desarrollo, se desaba probar la comunicación del sistema con una cámara Kinect. Puesto que no se dispuso de una hasta tiempo más tarde, se decidió comenzar las pruebas con una simulación de desarrollo propio.

A diferencia de la programación en ROS, para crear un simulador en gazebo es necesario crear archivos .launch, escritos en lenguaje de sintaxis marcada. Señalar que; aunque crear el modelo puede resultar bastante sencillo, suele consistir en descargar desde una base de datos desarrollada por Willow Garage, la adición de los topics de publicación y pautas de comportamiento es bastante compleja. Por ello, la estrategia consistió en crear un paquete propio que partiese del simulador realizado por Shadow del brazo completo, limitando el desarrollo a integrar Kinect como elemento adicional. No obstante, esta artimaña contó con más dificultades de las esperadas.

Debido a los complejos comportamientos que tuvo la empresa que definir para crear un simulador del robot, se creó una inimaginable cantidad de ficheros .launch de llamada recursiva que tuvieron que ser leídos y revisados para comprender el funcionamiento del simulador. En este punto, se recibió ayuda del equipo de Shadow (Toni.O y Armando), con motivo de contar con asesoramiento sobre el modo de configuración y programación de los ficheros .launch que componen el simulador (los archivos de gazebo se escriben con lengauje marcado XML y YAML). El principal problema surgía a la hora de crear los topics que genera un sensor de profundidad normal. No obstante, este problema se solventó rápidamente gracias a Toni.

El siguiente problema surgía a la hora de cargar el modelo. Gazebo, al tratar de lanzar el simulador mostraba el siguiente error:

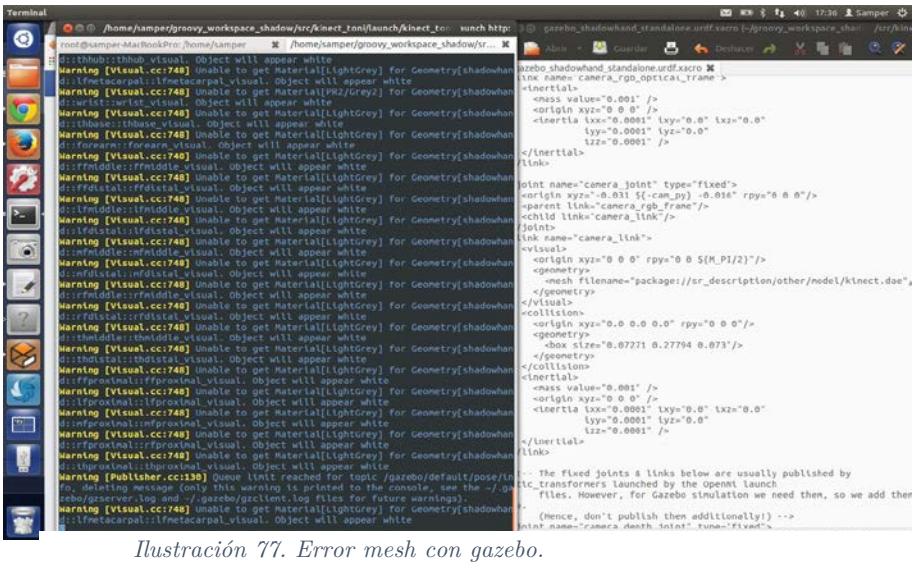


*Ilustración 77. Error mesh con gazebo.*

Tras varias pruebas, se dedujo que el fallo se encontraba en la etiqueta mesh que señala el modelo .dae del que se debe adquirir la configuración geométrica. Para solucionarlo se procedió a buscar todos los modelos disponibles en el equipo y comprobar cuál de ellos satisfacía mejor las necesidades. Tras estas modificaciones, se logró obtener un simulador conjunto del sistema. Sin embargo, el siguiente problema se encontró en la orientación de ambos objetos simulados.

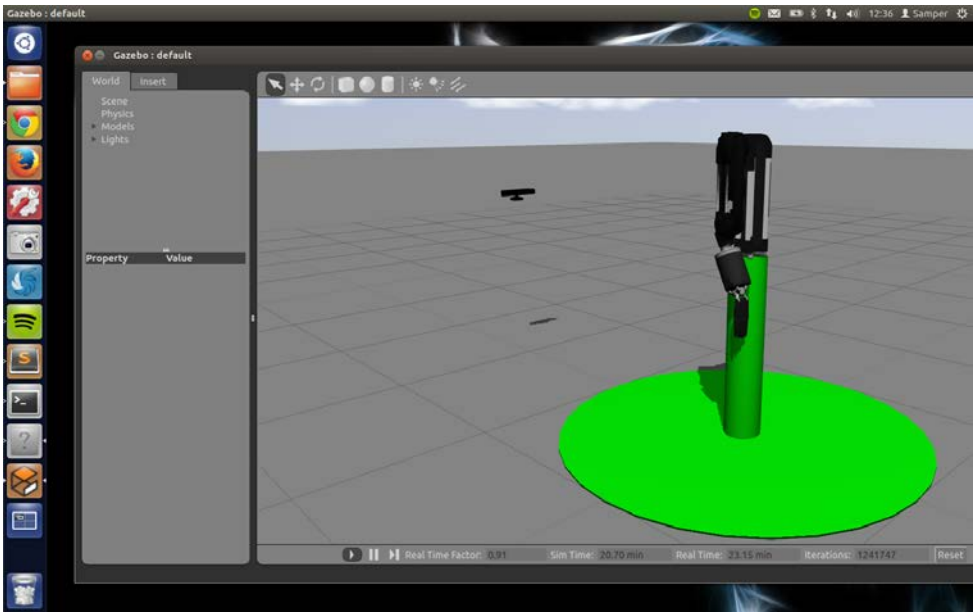



*Ilustración 78. Primera prueba conjunto de simulación.*

Se tiene que, por medio de los campos property node, de los distintos modelos, se determina la posición del elemento en el mundo creado por gazebo, un ejemplo puede observarse en la siguiente imagen:

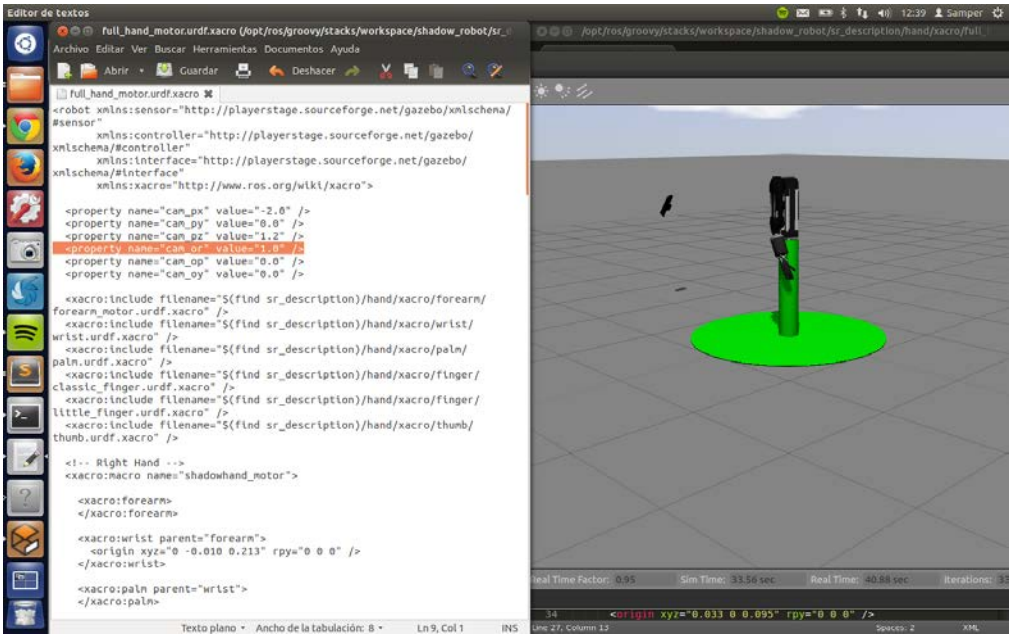



*Ilustración 79. Modificación parámetros de posición de gazebo,*

Finalmente, se consiguió arrancar el simulador con la orientación adecuada:

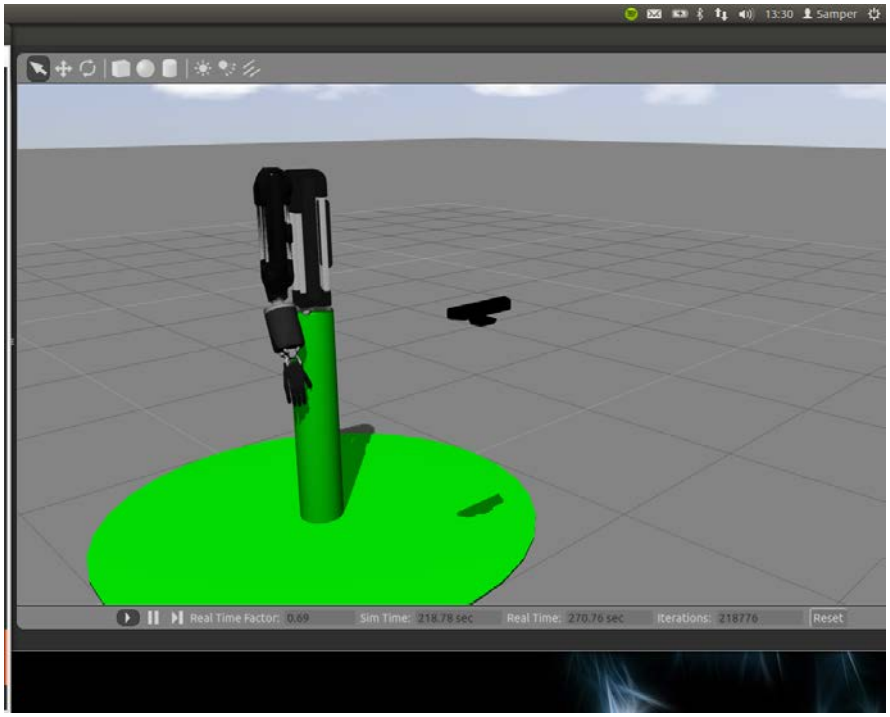



*Ilustración 80. Ejemplo de funcionamiento de gazebo.*

A partir de este momento, era posible usar el simulador como si de un conjunto de robot y sistema de visión artificial tradicional se tratase. Pudiéndose probar los topics publicados por el simulador de Kinect en rviz, siendo mínimas las diferencias con las proyecciones de una cámara real.

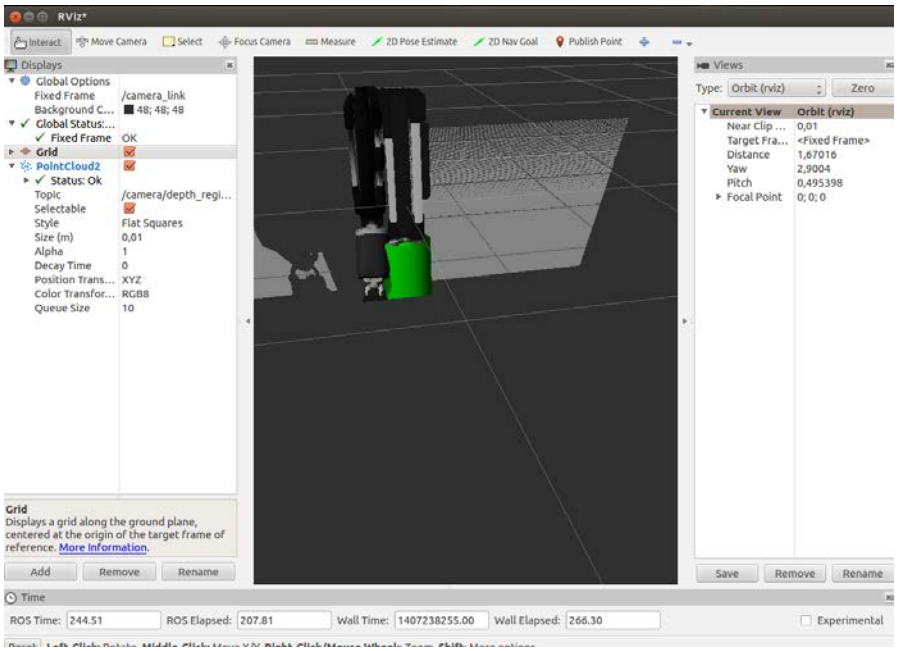
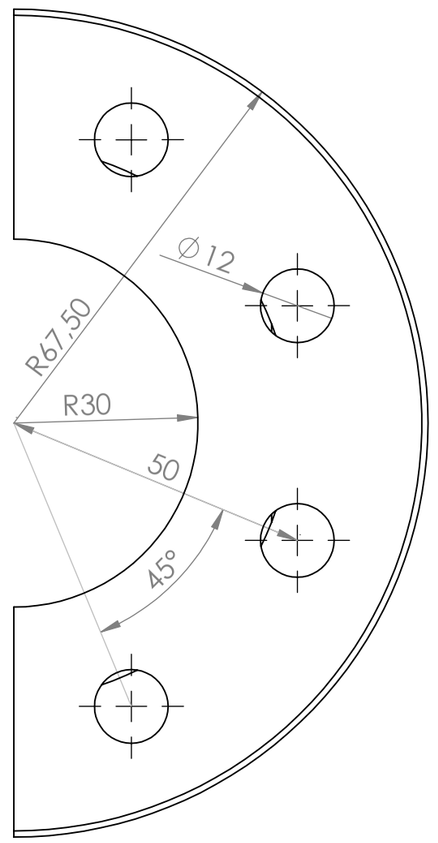


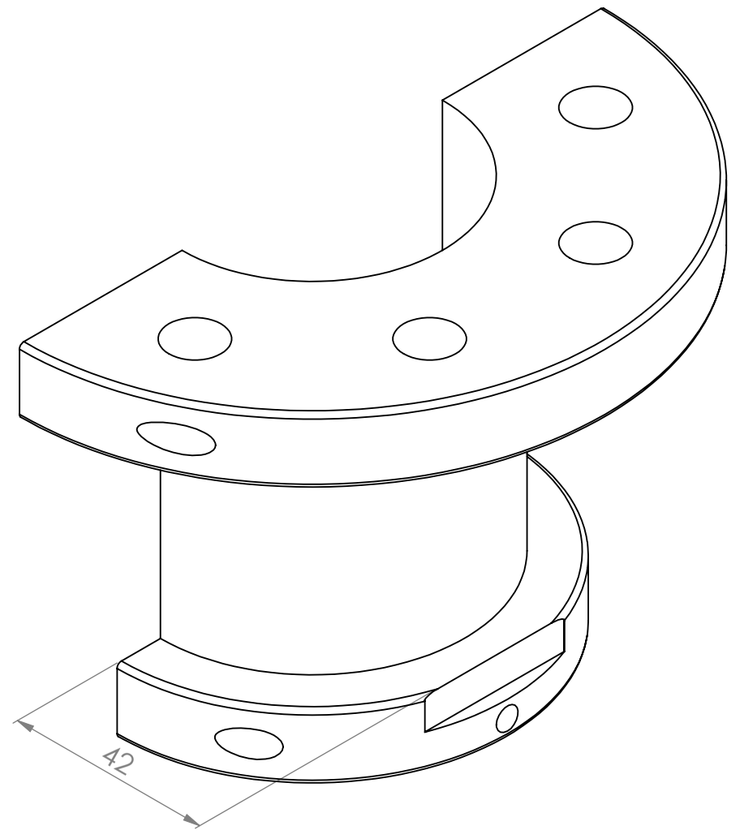

*Ilustración 81. Uso de los topics del kinect simulado en rviz.*

# Anexo 12. Planos de Acoples diseñados.

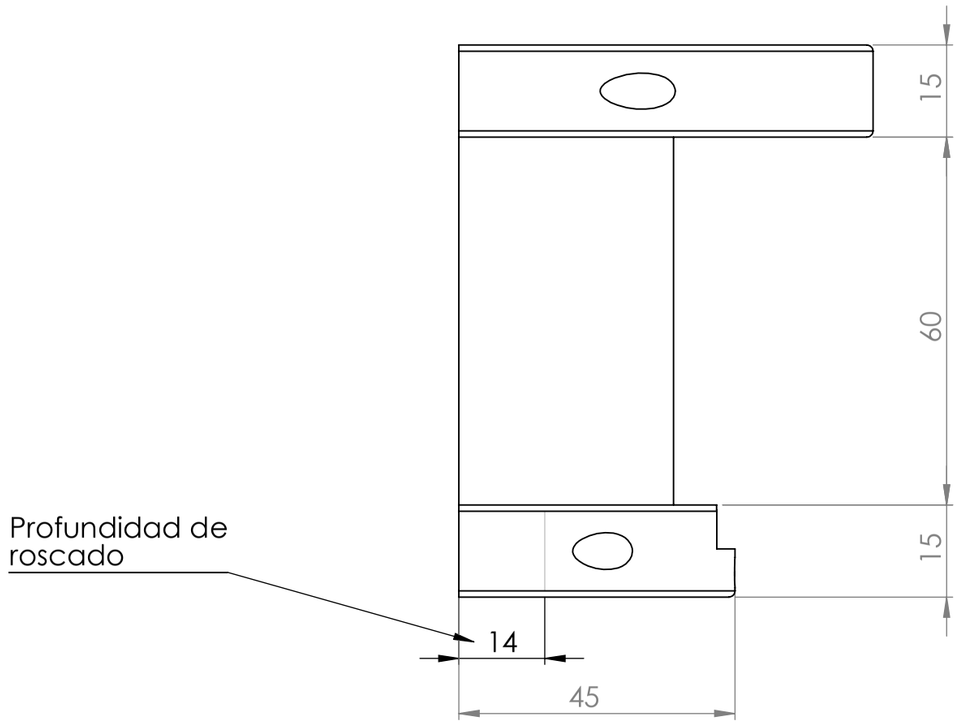
Todos los planos aquí realizados han sido diseñados en base a una idea común desarrollada junto a Joaquín Macanás Valera, poseyendo éste la autoría de los planos.

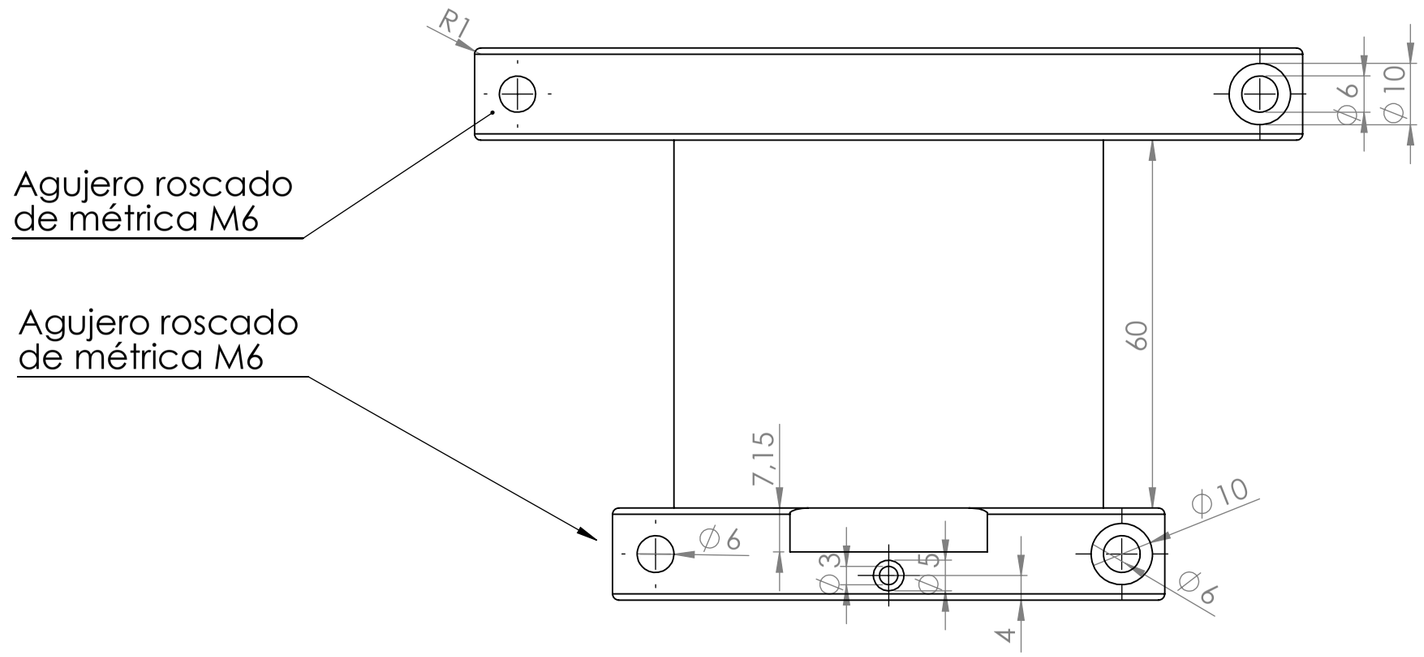
Universidad Politécnica de Cartagena



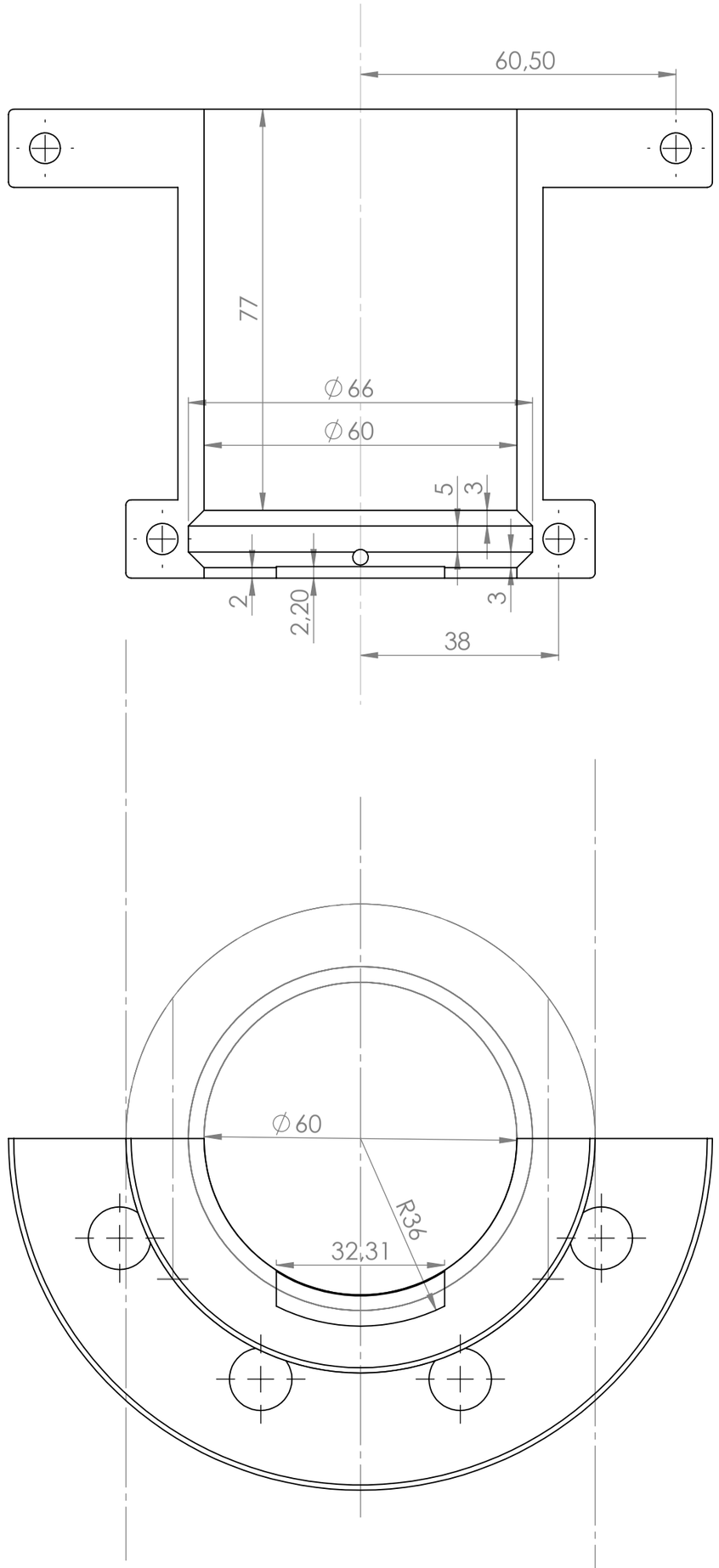

Todos los suavizados de bordes en la pieza son de radio = 1 mm

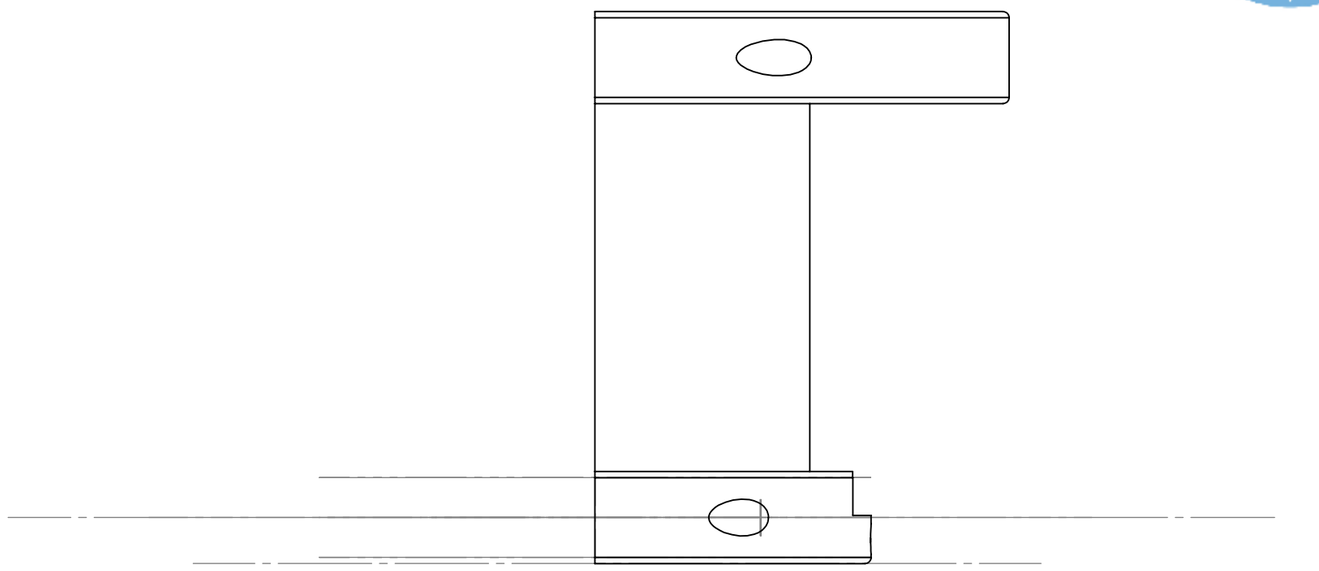


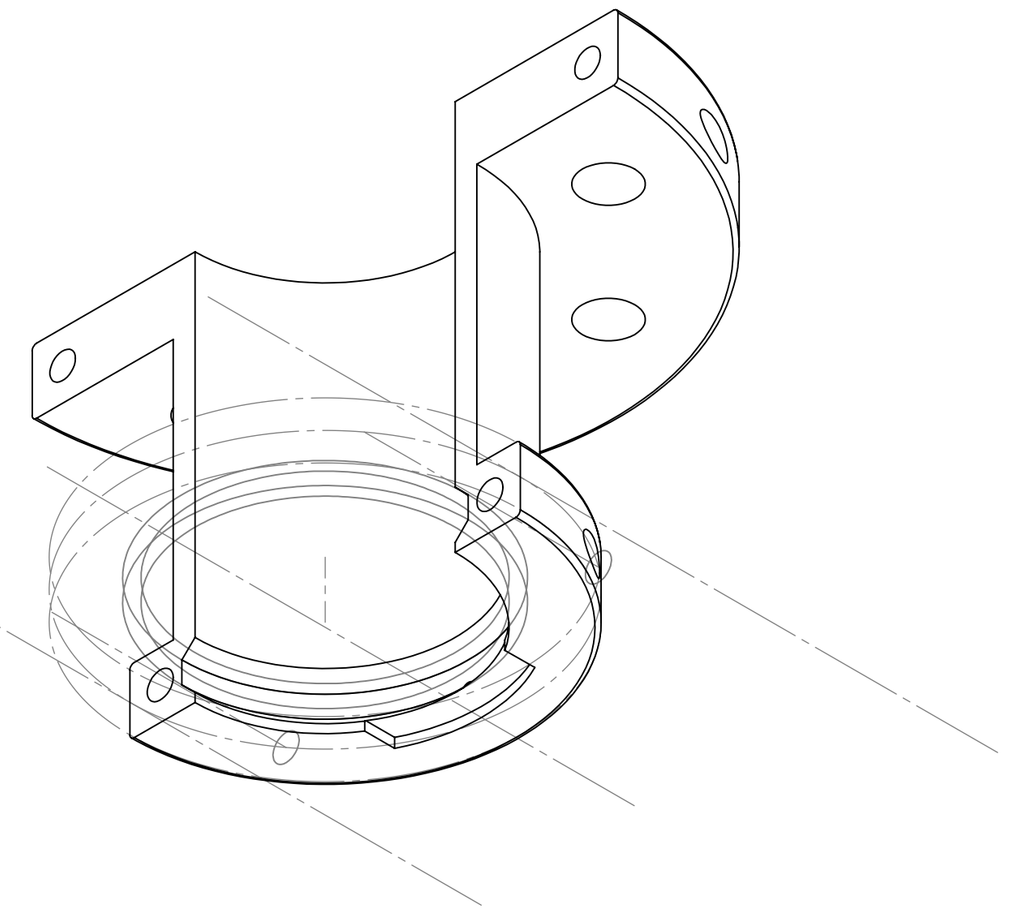




| | | | | | | | |
|---|---|---|---|---|---|---|---|
| SI NO SE INDICA LO CONTRARIO: LAS COTAS SE EXPRESAN EN MM<br>ACABADO SUPERFICIAL:<br>TOLERANCIAS:<br>LINEAL:<br>ANGULAR: | | | ACABADO: | | | REBARBAR Y ROMPER ARISTAS VIVAS | NO CAMBIE LA ESCALA / REVISIÓN |
| | | | | | | | Joaquín Macanás Valera |
| | NOMBRE | FIRMA | FECHA | | | | TÍTULO: |
| DIBUJ. | | | | | | | |
| VERIF. | | | | | | | |
| APROB. | | | | | | | |
| FABR. | | | | | | | |
| CALID. | | | | MATERIAL: | | | N.º DE DIBUJO: Acople brazo seccionado / A2 |
| | | | | PESO: | | | ESCALA:1:1 / HOJA 1 DE 2 |

Universidad Politécnica de Cartagena

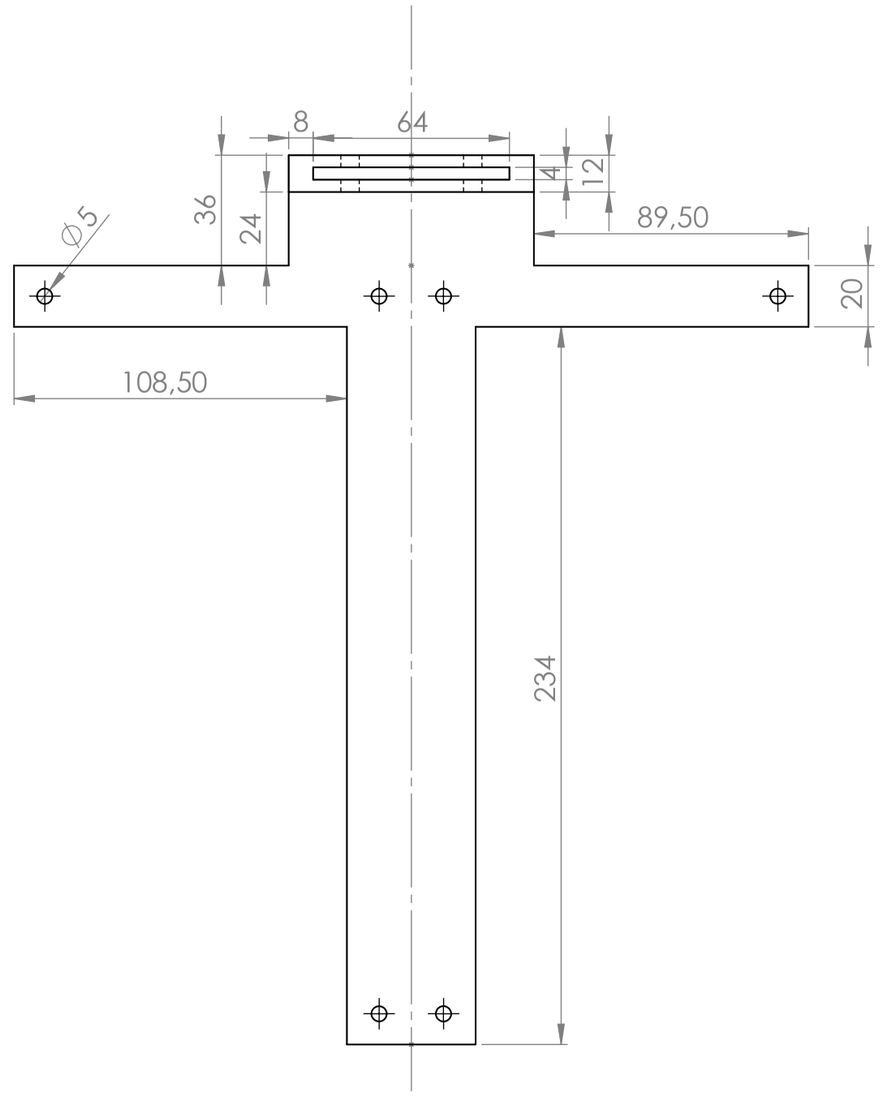


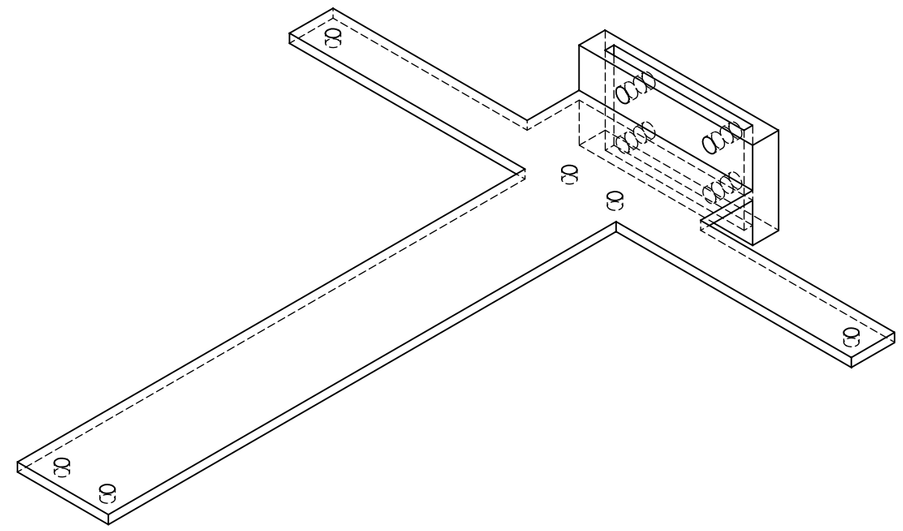



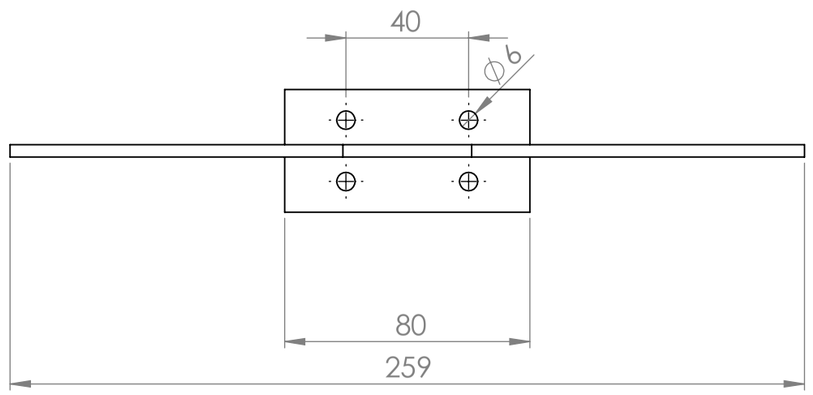


| | | | | | | | |
|---|---|---|---|---|---|---|---|
| SI NO SE INDICA LO CONTRARIO: LAS COTAS SE EXPRESAN EN MM<br>ACABADO SUPERFICIAL:<br>TOLERANCIAS:<br>LINEAL:<br>ANGULAR: | | ACABADO: | | | | REBARBAR Y ROMPER ARISTAS VIVAS | NO CAMBIE LA ESCALA / REVISIÓN |
| | | | | | | | Joaquín Macanás Valera |
| | NOMBRE | FIRMA | FECHA | | | | TÍTULO: |
| DIBUJ. | | | | | | | |
| VERIF. | | | | | | | |
| APROB. | | | | | | | |
| FABR. | | | | | | | |
| CALID. | | | | MATERIAL: | | | N.º DE DIBUJO: Acople brazo seccionado — A2 |
| | | | | PESO: | | | ESCALA:1:1 — HOJA 2 DE 2 |

Universidad Politécnica de Cartagena

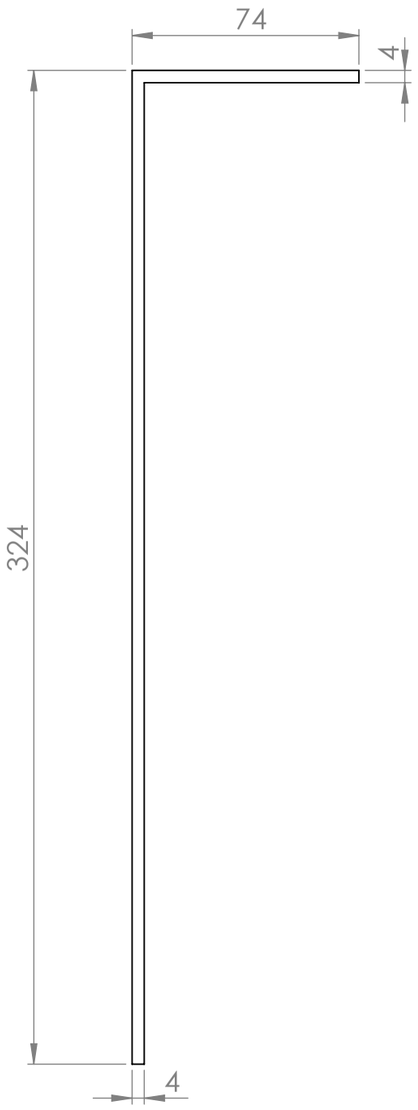


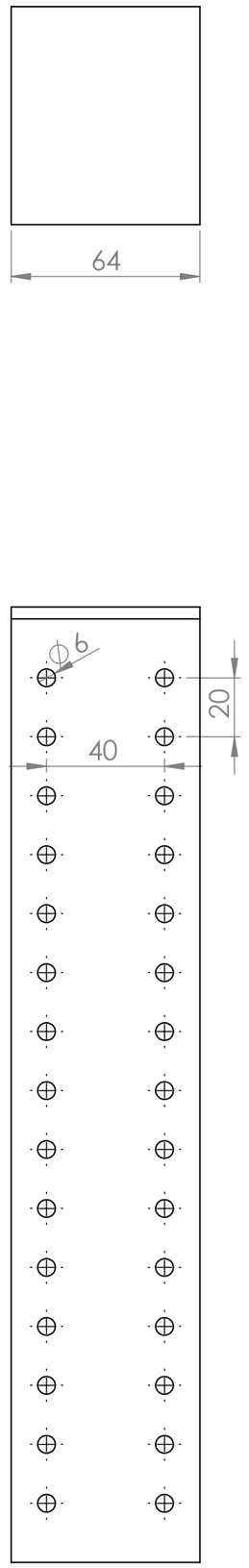



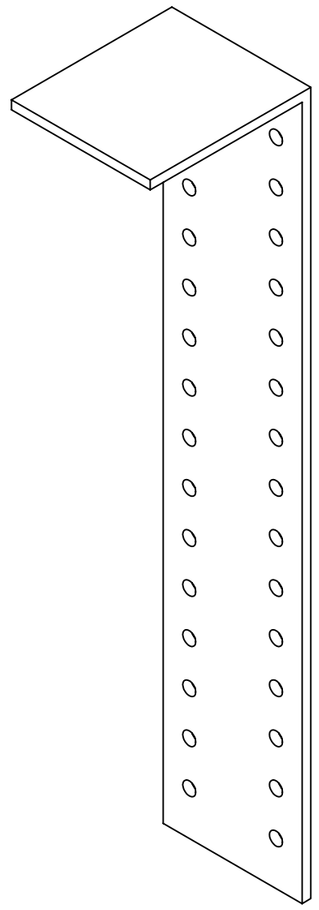




| | | | | | | |
|---|---|---|---|---|---|---|
| SI NO SE INDICA LO CONTRARIO: LAS COTAS SE EXPRESAN EN MM<br>ACABADO SUPERFICIAL:<br>TOLERANCIAS:<br>LINEAL:<br>ANGULAR: | | ACABADO: | | | REBARBAR Y ROMPER ARISTAS VIVAS | NO CAMBIE LA ESCALA / REVISIÓN |
| | | | | | | Joaquín Macanás Valera |
| | NOMBRE | FIRMA | FECHA | | | TÍTULO: |
| DIBUJ. | | | | | | Soporte cámara horizontal |
| VERIF. | | | | | | |
| APROB. | | | | | | |
| FABR. | | | | | | |
| CALID. | | | | MATERIAL: | | N.º DE DIBUJO / A2 |
| | | | | Acero inoxidable | | |
| | | | | PESO: | | ESCALA:1:1 / HOJA 1 DE 1 |

Universidad Politécnica de Cartagena

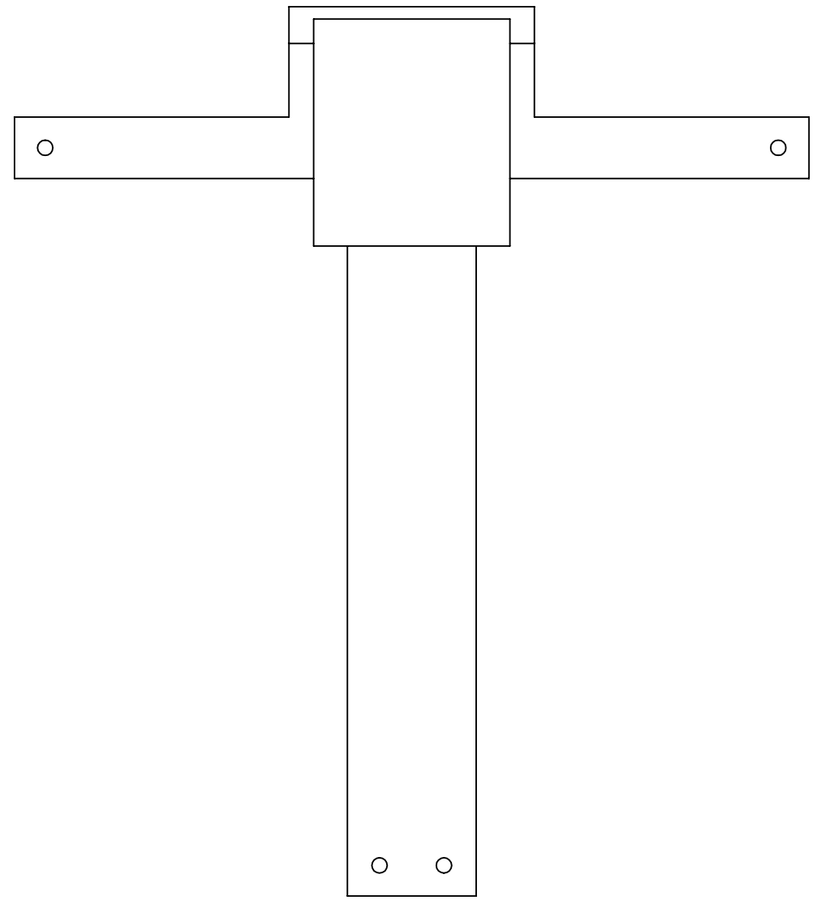


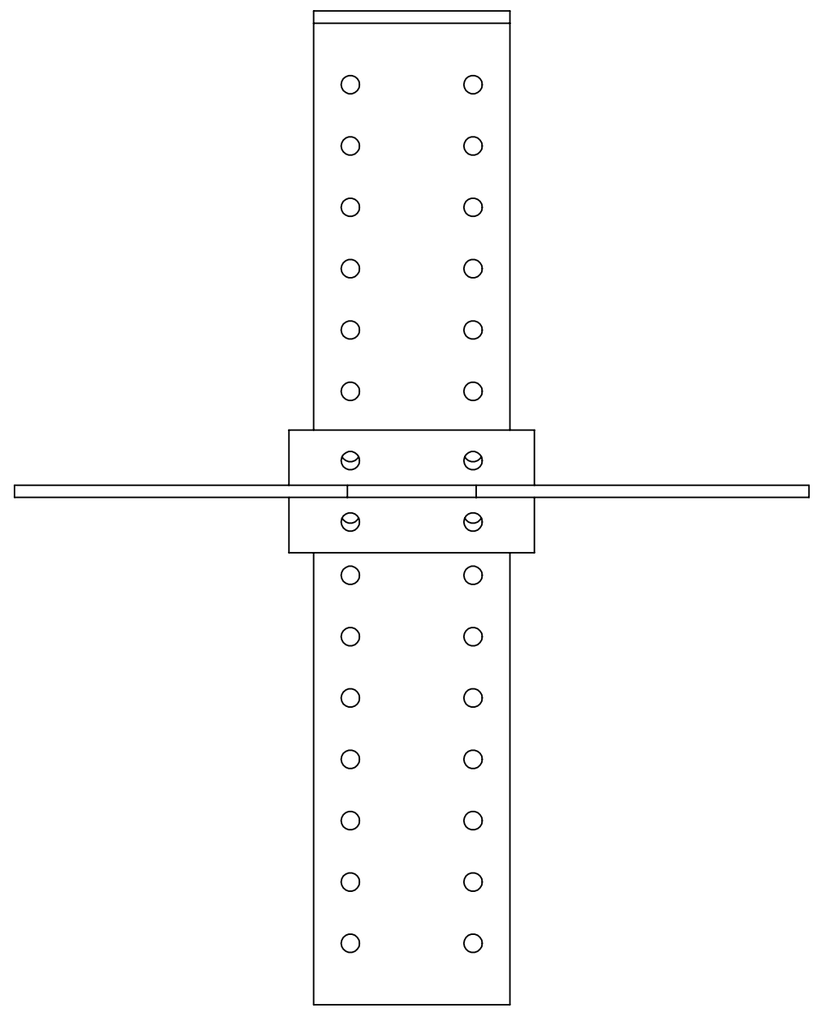



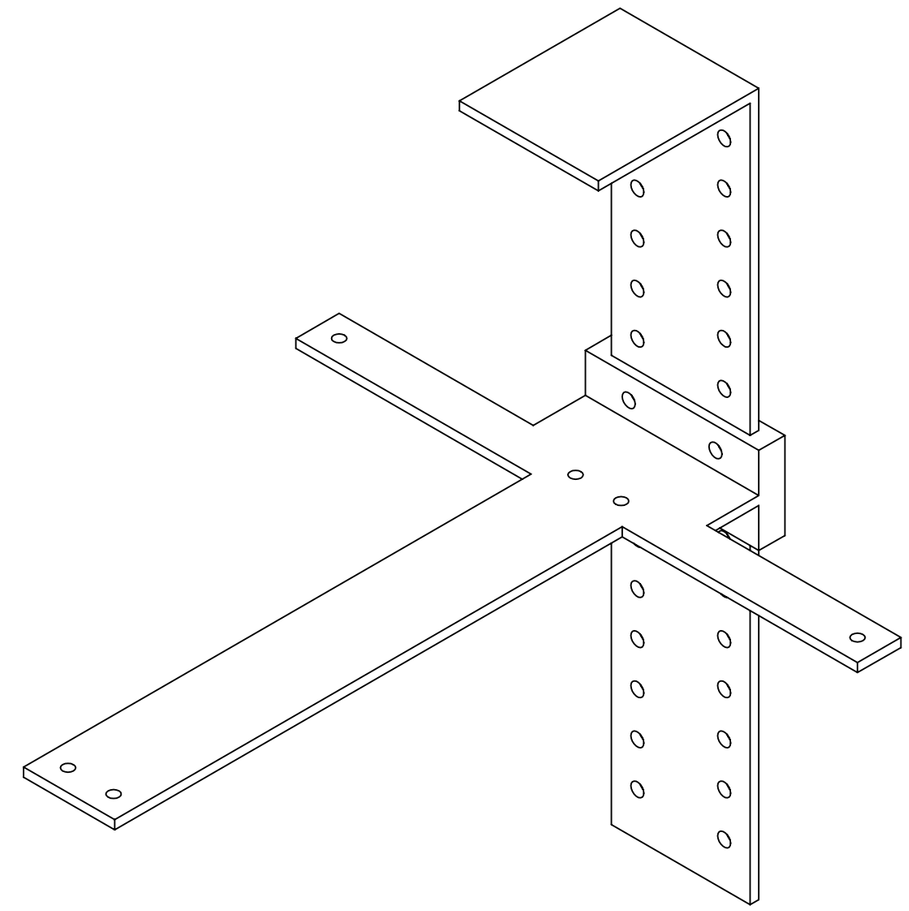



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| SI NO SE INDICA LO CONTRARIO: LAS COTAS SE EXPRESAN EN MM ACABADO SUPERFICIAL: TOLERANCIAS: LINEAL: ANGULAR: | | | ACABADO: | | | REBARBAR Y ROMPER ARISTAS VIVAS | NO CAMBIE LA ESCALA / REVISIÓN |
| | NOMBRE | FIRMA | FECHA | | | | Joaquín Macanás Valera |
| DIBUJ. | | | | | | | TÍTULO: Soporte cámara vertical |
| VERIF. | | | | | | | |
| APROB. | | | | | | | |
| FABR. | | | | | | | |
| CALID. | | | | MATERIAL: Acero inoxidable | | | N.º DE DIBUJO: A2 |
| PESO: | | | | | | | ESCALA:1:1 / HOJA 1 DE 1 |

Universidad
Politécnica
de Cartagena







| | | | | | | | |
|---|---|---|---|---|---|---|---|
| SI NO SE INDICA LO CONTRARIO: LAS COTAS SE EXPRESAN EN MM<br>ACABADO SUPERFICIAL:<br>TOLERANCIAS:<br>LINEAL:<br>ANGULAR: | | | ACABADO: | | | REBARBAR Y ROMPER ARISTAS VIVAS | NO CAMBIE LA ESCALA / REVISIÓN |
| | NOMBRE | FIRMA | FECHA | | | | Joaquín Macanás Valera |
| DIBUJ. | | | | | | | TÍTULO: |
| VERIF. | | | | | | | Ensamblaje soporte cámara |
| APROB. | | | | | | | |
| FABR. | | | | | | | |
| CALID. | | | | MATERIAL: | | | N.º DE DIBUJO: A2 |
| | | | | | | | |
| | | | | | | | |
| | | | | PESO: | | | ESCALA:1:1 / HOJA 1 DE 1 |

# Índice de Ilustraciones.

# Bibliografía


1] J. Santaella., «Tabla Comparativa OS.,» [En línea]. Available: HTTP://WWWJOSEJUANSANTAELLA.BLOGSPOT.COM.ES/2012/10/TABLA-COMPARATIVA.HTML.

2] «Blog de Comparación de distintos Sistemas Operativos.,» [En línea]. Available: 2HTTP://1.BP.BLOGSPOT.COM/_TOH1RMOORVI/R5JMN0KZMRI/AAAAAAAAAEG/WNZS6C3J3OO/S1600/COMPARATIVA+SO.JPG .

3] M. Linux, «El rincón de Linux para hispanohablantes,» Muy Linux, [En línea]. Available: http://www.linux-es.org/distribuciones.

4] C. D. &. W. D.Smart, «Middleware for Robots?,» de *AAAI Symposium Workshop on Intelligent and Distributed Embedded Systems*, Stanford, CA, 2002.

5] a. D. J., *Middleware. In Urban,,* vol. 2012, Bakken: Encyclopedia of Distributed Computing, 2001, p. 15.

6] M. M. Simone Ceriani, *Middleware in robotics,* INTERNAL REPORT FOR "ADVANCED METHODS OF INFORMATION TECHNOLOGY FOR AUTONOMOUS ROBOTICS.

7] «Yet Another Robot Platform,» [En línea]. Available: http://wiki.icub.org/yarpdoc/what_is_yarp.html.

8] A. E. a. T. Sobh, «Robotics Middleware: A Comprehensive Literature Survey and Attribute-Based Bibliography,» *Journal of Robotics,* vol. 2012, nº Article ID 959013, p. 15, 2012.

9] F. M. A. V. P. F. y. C. A. González, Técnicas y Algoritmos Básicos de Visión Artificial., Universidad de La Rioja: Universidad de La Rioja, 2006.

R. W. &. H. ELIAS, *SHADOW CARTAGENA HAND,* Liverpool,
10] UK, 2011.

The Shadow Robot Company, Ltd., *Shadow C6M2 Hand - User*
11] *Manual.,* Liverpool, UK., 2009.

The Shadow Robot Company, Ltd., *Robot Control Software API,*
12] Liverpool, UK., 2009.

L. The Shadow Robot Company, *Shadow Robot Software System,*
13] Liverpool, UK., 2009.

IDS, «IDS It's so easy!,» [En línea]. Available: http://es.ids-
14] imaging.com/store/produkte/kameras/usb-2-0-kameras/ueye-se.html.

W. Garage, «Página Oficial de ROS,» Willow Garage, [En línea].
15] Available: http://wikiros.org.

V. G. Díaz, «Desarrollo Robótico. Robot Operating System.,» 2013.
16] [En línea]. Available: HTTP://ES.SLIDESHARE.NET/VICEGD/DESARROLLO-ROBTICO-ROBOT-OPERATING-SYSTEM-ROS.

W. Garage, «Página Oficial de Willow Garage.,» Willow Garage,
17] 2008. [En línea]. Available: https://www.willowgarage.com.

Wikipedia, «Robot Operating System,» Wikipedia, 2014. [En línea].
18] Available: http://en.wikipedia.org/wiki/Robot_Operating_System.

PointCloud, «PCL: Point Cloud Library.,» PCL, [En línea].
19] Available: HTTP://POINTCLOUDS.ORG.

PCL, «Tutoriales PCL,» PCL, [En línea]. Available:
20] http://pointclouds.org/documentation/tutorials/.

D. F. Tombari, «SESSION IV: 3D OBJECT RECOGNITION IN
21] CLUTTER WITH THE POINT CLOUD LIBRARY.,» de *ROS-RM*, Universidad de Alicante, 2014.

P. D. &. H.M.Deitel, Cómo Programar C++, Sexta edición. 22] Pearson-Prentice Hall., 2008.

M. Ibáñez, *LABORATORIO DE ARQUITECTURA DE* 23] *ORDENADORES. PRÁCTICA 1: MAKEFILE,* Universidad Carlos III de Madrid.: DEPARTAMENTO DE INGENIERÍA TELEMÁTICA, 2012.

V. A. G. BARBONE, «Herramienta Make.,» 2010. [En línea]. 24] Available: HTTP://IIE.FING.EDU.UY/~VAGONBAR/GCC-MAKE/GCC.

G. A. GARCÍA, *MAKE. UN PROGRAMA PARA CONTROLAR* 25] *LA RECOMPILACIÓN..*

«Robots compatibles con ROS.,» 2014. [En línea]. Available: 26] HTTP://WIKI.ROS.ORG/ROBOTS.

ROS, «ROS Ubuntu Installation,» 2013-2014. [En línea]. Available: 27] HTTP://WIKI.ROS.ORG/HYDRO/INSTALLATION/UBUNTU.

S. R. Company, «ROS- Shadow Robot' section,» [En línea]. 28] Available: HTTP://WIKI.ROS.ORG/SHADOW_ROBOT.

ROS, «ROS's Tutorials,» 2014. [En línea]. Available: 29] HTTP://WIKI.ROS.ORG/ROS/TUTORIALS.

ROS, «ROS. Creating a Package Tutorial,» [En línea]. Available: 30] http://wiki.ros.org/ROS/Tutorials/CreatingPackage.

ROS, «ROS. Understading Topics.,» [En línea]. Available: 31] http://wiki.ros.org/ROS/Tutorials/UnderstandingTopics.

ROS, «ROS. Using RQT Console And ROSLaunch,» [En línea]. 32] Available: HTTP://WIKI.ROS.ORG/ROS/TUTORIALS/USINGRQTCONSOLEROSLAUNCH.

GAZEBO, «GAZEBO Web Site,» [En línea]. Available:
33] HTTP://WWW.GAZEBOSIM.ORG.

Shadow Company, «ROS. Extra Documentation,» [En línea].
34] Available:
HTTP://WIKI.ROS.ORG/SHADOW_ROBOT?ACTION=ATTACHFI
LE&DO=VIEW&TARGET=ANNOTATED_HAND.PNG.

Robótica Unileon, «Robótica Unileon. PCL Tutorial.,» [En línea].
35] Available:
HTTP://ROBOTICA.UNILEON.ES/MEDIAWIKI/INDEX.PHP/PCL/
OPENNI_TUTORIAL_1:_INSTALLING_AND_TESTING.

HDF5, «What is HDF5 ?,» [En línea]. Available:
36] http://www.hdfgroup.org/HDF5/.

beej, «Client-Server Background,» [En línea]. Available:
37] http://beej.us/guide/bgnet/output/html/multipage/clientserver.html.

C. Roxas, «Sockets en C de Unix/Linux,» [En línea]. Available:
38] http://beej.us/guide/bgnet/output/html/multipage/clientserver.html.

Sascha Nitsch Unternehmensberatung UG, «Linux Howtos: C/C++
39] -> Sockets Tutorial,» [En línea]. Available:
http://www.linuxhowtos.org/C_C++/socket.htm.

A. Pargelas, «Programación Básica de Sockets en Unix para
40] Novatos,» [En línea]. Available: http://es.tldp.org/Tutoriales/PROG-
SOCKETS/prog-sockets.html.

J. G. d. J. &. co, Aprenda Linux como si estuviera en primero.,
41] Escuela Superior de Ingenieros Industriales de San Sebastián..

G. Bradsky, Learning OpenCV [computer vision with the OpenCV
42] Library], O'Reilly, 2008.

D. L. Baggio, Mastering OpenCV with practical computer vision
43] projects, Packt Pub, 2012.

M. Lutz, Learning Python 2nd ed. Covers Python 2.3, O'Reilly,
44] 2003.

Micosoft, «Kinect for Windows Sensor,» Microsoft, [En línea].
45] Available: http://msdn.microsoft.com/en-us/library/hh855355.aspx.